目录

第 5 章　图像平滑

第 6 章　图像增强

第 7 章　特征选择与描述

第 8 章　彩色变换

第 9 章　彩色分割

附录 专业版 Imgc 的界面和程序

第 1 章

走进图像处理世界

1.1 基于计算机的图像处理

在这个高度信息化的社会中，图形图像和计算机在人们的生活中显得越来越重要。于是用计算机处理图像的技术也得到了迅速的发展和普及。

图像(image)有各种各样的形式，图 1-1 对各种图像进行了归纳。根据处理图像的种类、处理结果的精度、处理速度的不同，有必要选择不同的处理方法。

现在电视机的显像管已经被用在个人计算机的显示器中，我们也可以在个人计算机中观看电视节目了。随着图像处理技术和计算机技术的发展，摄取图像的环境也越来越多，图 1-2 给出了一些示例。当然，随着技术的进步，模拟世界的电视机已经吸收了数字化技术和计算机技术，开始了全数字化的数字电视广播。打印机也从原来只能处理文字和灰度图像发展到能够处理彩色动态图像了。计算机图形学(computer graphics)和图像处理(image processing)是随着计算机技术的发展兴起的两种技术，但是它们相互作用，共同成长，现在已经很难把它们严格区分开来了。

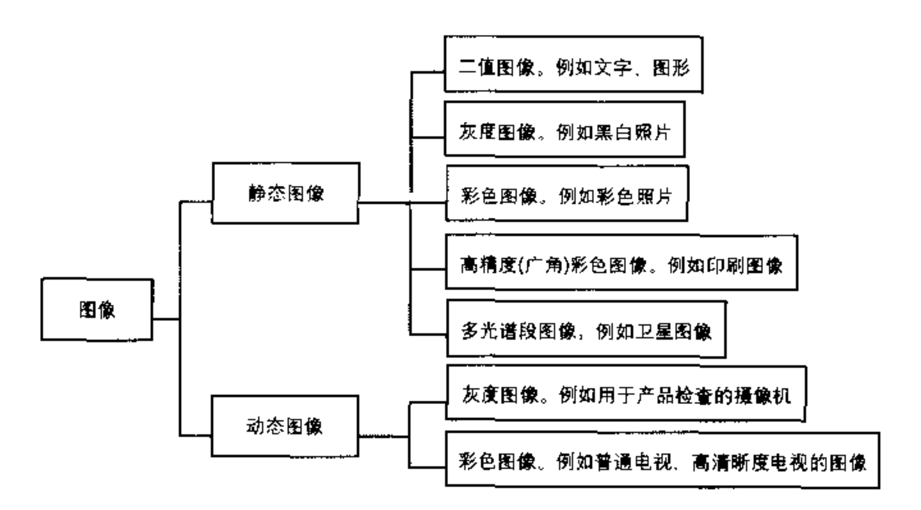


图 1-1



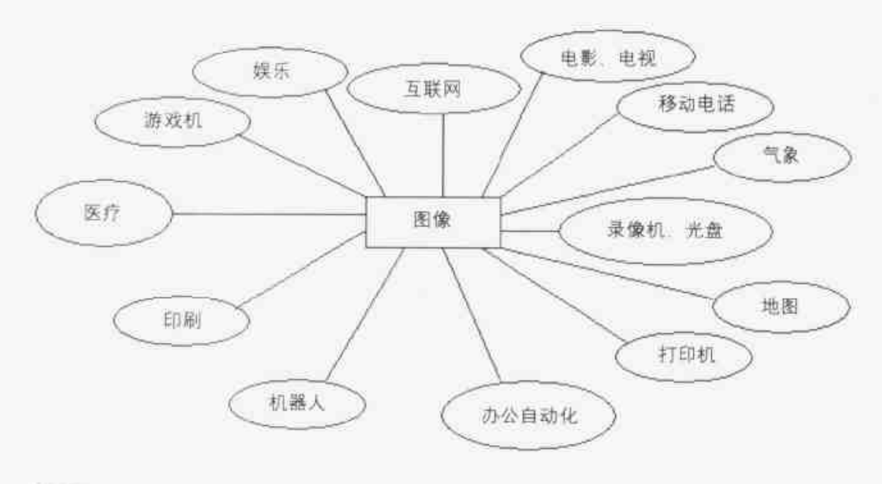


图 1-2

1.2 各种图像处理的特点

电视机中的特殊效果、自动售货机中纸币的读取、邮政编码的自动识别等都用到了图像处理技术。图像处理应用越来越广泛，例如，在医院里很早就使用 X 射线与显微镜照片来诊断疾病，现在采用计算机处理图像已经成为疾病诊断的重要方法。

另外，把人体内的状态进行图像化的特殊图像处理装置也在疾病诊断中使用。例如 MRI（magnetic resonance imaging，核磁共振成像）和 CT（computed tomography，计算机断层摄影），图 1-3 是 MRI 图像与 CT 图像的融合切片图，左侧的 MRI 图像显示软组织形态，而右侧的 CT 图像显示骨骼的结构。以前不解剖就不可能知道的脑中状态，现在由图像处理就可以近似模拟出来了。这个方法是划时代的，许多医学书籍甚至已经被 MRI 图像与 CT 图像重新改写。

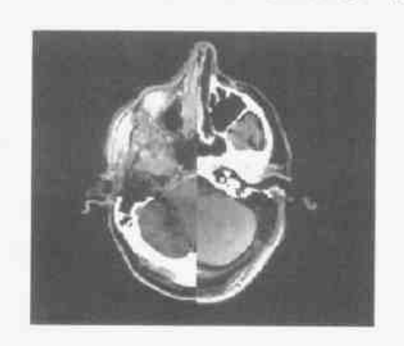


图 1-3



图像处理的对象非常广泛，图像处理技术也应用于越来越多的领域中，如表 1–1 所示。

表 1-1 各种各样的图像

应用领域	图像示例
办公室	文本 画面、商标
医学	X 射线图像、超声波图像、显微镜图像、MRI 图像、CT 图像
遥感	地球资源卫星（landsat）图像、气象卫星图像、航空图片、地图
工厂	IC 模式图、工业摄像机拍摄的图像
电影电视	各种照片（风景、人物） 计算机图形学图像、电影电视、摄像机拍摄的图像
其他	民俗资料、用于研究的图像、指纹、印刷图像

下面给出几种实际应用在我们身边的图像处理。

1．办公室中的图像处理

对于文本和画面上的应用，主要是对黑白二值图像进行处理。例如，对文本图像上文字的自动判别、自动识别手写画面等。

2．医学中的图像处理

在医学领域中很早就开始处理 X 射线照片和显微镜中的成像等大量图像，利用图像处理技术进行了染色体的分析与细胞的自动分类等研究，这是图像处理最先进的领域。

另外，由于近年来科学技术的快速发展，出现了可以把无法看见的世界图像化的方法，前面说明的 MRI 和 CT 就是代表性的实例，现在还出现了使用超声波来观察胎内婴儿动作的装置，

3．遥感图像处理

这是处理从人造卫星拍摄的图像，使资源信息和气象信息等图像化的方法。主要应用在农业、渔业、环境污染调查、城市规划等方面。

4．工业中的图像处理

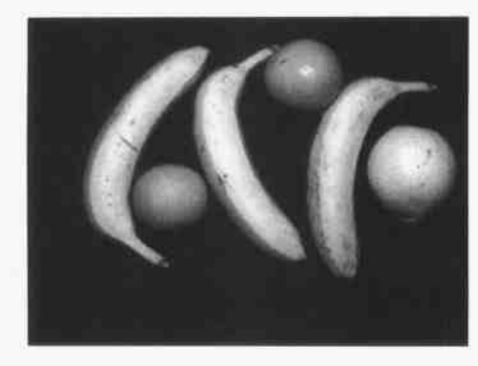
在工厂自动化中也使用各种图像处理。主要用在缺欠品的自动检查、产业用机器人的视觉等方面。为了在安装和生产线上使用，对于从摄像机输入的图像，有必要进行高速简单的处理。

5．电视、电影中的图像处理

在电视、电影领域中，作为特殊效果来使用的情况很多，如在新节目制作方法的开发中利用图像变形或者如图 1–4 所示的图像合成等，其中（a）是前景图像，（b）是背景图像，（c）是图像合成。



(a) (b)
(c)





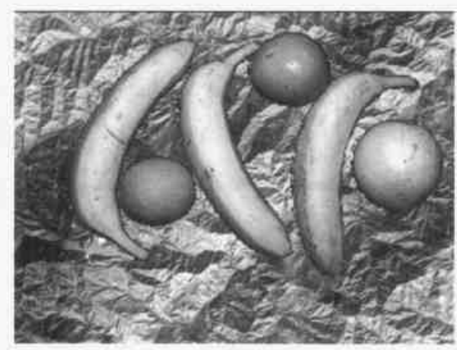


图 1-4

6. 其他

最近，出现了识别指纹图像或者眼膜纹图像的电子钥匙等。

由于个人计算机技术的快速发展，图像处理越来越贴近我们的生活。以前只有高级的计算机才能处理的图像，现在用个人计算机便能处理。而且，由于数字通信技术的进步、互联网的普及，许多图像和图像处理程序都能很容易找到，渐渐营造出个人也能很容易体验图像处理的环境。

1.3 数字图像处理的特征

现在说到图像处理，就意味着基于计算机的图像处理，但是也有使用光学系统的模拟图像处理（analog image processing）。例如，在照片摄影中装上遮光片拍摄柔和的气氛、打开快门拍摄流星的运动。数字图像处理（digital image processing）就是通过在计算机上进行计算来实现与上述相同的处理。数字图像处理的优点可总结为以下几点。



- 处理正确、具有再现性。

  由于通过计算机进行处理，正确性不必说了，而且同样的程序即使数次运行，也能得到同样的结果，具有再现性。

- 容易控制。

  通过程序能够自由设定及变更控制用的各种参数。

- 处理的多样性。

  处理都是由程序进行的，只要变更一下程序，就能够实现各种各样的处理。另外，能够组合程序，自己开发新程序。

另一方面，也有其缺点。

- 数据量大。

  图像数据数字化后输入到计算机内，例如，把 17 英寸电视屏幕大小的图像按各个颜色数字化后输入计算机内，就相当于水平 700 像素 × 垂直 500 像素 ×3 个单位色约为 1 MB 的数据。这相当于一张软盘的容量。

- 费时。

  由于数据量大，处理图像所花费的时间多。如果处理 1 个像素花 1 毫秒，处理 700 × 500 像素的数据就要花 350 秒。



第 2 章

图像处理的基础知识

2.1 图像处理的硬件构成

图像处理需要什么设备？图 2−1 给出了一般图像处理系统的构成：用于图像获取的图像输入装置、用于图像存储的存储器以及进行图像处理的中央处理器（CPU）等。其实，一台个人计算机（personal computer）加上扫描仪（scanner）就构成了一种图像处理系统。下面将详细介绍构成图像处理系统的各部分。

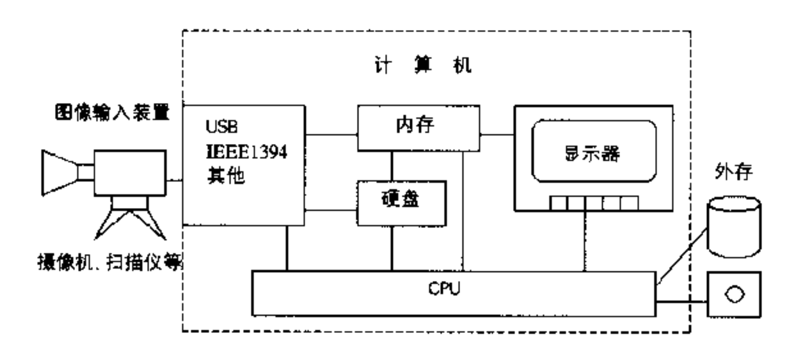


图 2-1

1．图像输入装置（image input device）

传统的图像获取（image acquisition）方式是由摄像机（video camera）、扫描仪（scanner）等装置，通过图像采集卡（image grabber），把模拟图像数字化后输入到计算机中。现在，由于数码技术的快速发展，图像输入装置也发生了革命性的变化，可以由数码相机（digital still camera），数码扫描仪（digital scanner），VCD，DVD 播放装置等所有具有 IEEE1394 接口或者 USB 接口的数码图像设备直接把图像输入到计算机中。计算机的主板上都有 USB 接口。大多数便携式计算机除了 USB 接口之外，还带有 IEEE1394 接口。台式计算机在用 IEEE1394 接口的数码图像装置进行图像输入时，目前需要另配一枚 IEEE1394 图像采集卡，但是最近台式计算机的主板上也开始附加 IEEE1394 接口了。

2．内存（或称frame memory，帧存储器）

这是存储图像的地方。通常被存储的图像可在显示器上适时地显示出来。

一般的图像处理系统需要用 8 位来处理一种色彩，所以总共是 24 位，约 1677 万种的彩色（color）。像素数依处理的图像的不同而不同，早期经常采用 256 × 256 像素与 512 × 512 像素的图像。后来，大多处理从摄像机输入的 640 × 480 像素的图像了。



3．显示器（display）

显示器是图像和命令的显示窗口。现在，个人计算机的显示器可以选择多种像素及色彩的表示方式，从 640×480 像素的 256 色到 1600×1200 像素以及更高像素的 24 位的真彩色（true color）。

4．CPU（central processing unit）

计算机的核心部位。CPU 的发展是非常迅速的，它主要用来对图像进行各种各样的处理。

5．图像存储部件（image storage device）

数字化的图像数据与计算机的程序数据相同，被存储在个人计算机的硬盘或软盘中。也就是说，个人计算机将图像保存在硬盘或软盘中，通过计算机的处理后，再将图像保存在硬盘或软盘中以备使用。现在也能用扫描仪从通常的照片或书刊把图像输入到计算机中，也可以通过数码相机和摄像机来输入图像。另外，还可以通过市场上出售的存有图像的光盘或者通过因特网下载来获取图像。

所谓昂贵的图像处理装置，只不过是具有高性能的图像输入装置，具有大容量的图像存储装置，或者具有图像处理专用的硬件而已。但是，就一般图像处理而言，个人计算机已经足够了。

2.2 数字图像与 Visual C++

照片或者画面之类的图像是怎样输入到计算机中的呢？图像在计算机中按图 2-2 所示被分成像素（pixel），各个像素的灰度值（gray value，或称 value，浓淡值）被整数化（或称数字化，digitization）。

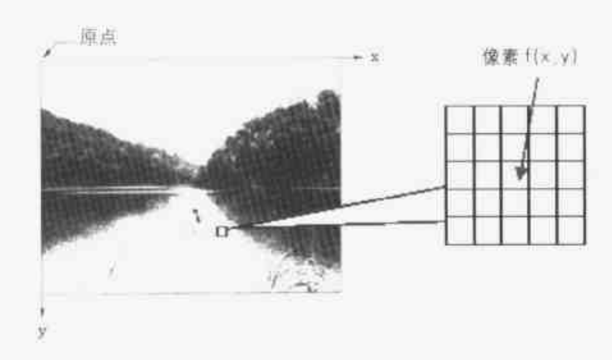


图 2-2



图 2–3 显示了一个简单的实例。这是空白图像与名画《蒙娜丽莎》的放大图，放大后可以看见图中的各个小方块即像素，该图是由 64×64 像素构成的。计算机的软件可以由 Basic、Fortran、C 等语言来记述，该书的图像处理程序是用 C 来记述的，图像的表示、对话框界面等用 Visual C++来表述，所以这本书以 Visual C++为平台揭示图像处理程序、界面程序的制作方法。



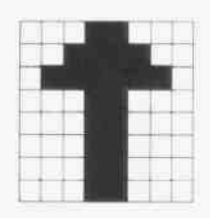
图 2-3

下面说明图像数据的排列方法。通常把数字图像左上角的像素作为位于(0,0)的像素考虑，从那个像素起在水平方向上的数值为 i、在垂直方向上为 j 的位置(i,j)的像素的输入灰度值用数组 image_in[j][i]或数据指针*(image_in + j*XSIZE + i) 表示，输出的灰度值用 image_out[j][i]或*(image_out + j*XSIZE + i)表示，XSIZE 为图像的宽度。

下面试做一个最简单的图像处理。如图 2–4 所示，把图像(a)倒转过来成为图像(c)。图像(a)的数据分布情况如(b)所示。用 C 语言编写程序如图 2–4(d)所示，从输入数组最后一行起，逐行将数据代入输出数组即可。输出数组的表示结果即为图像(c)。只要按要求做一些改变就可实现各种各样的图像处理了。

上述是灰度图像（gray scale image）数据的处理情况，对于彩色图像（color image），需要分别对彩色的红（Red，R）、绿（Green，G）、蓝（Blue，B）三个分量进行处理。

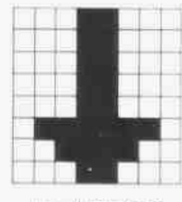




(a) 原始图像

```
BYTE *image_in =
{ {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 0, 0, 0, 0, 1, 1},
  {1, 0, 0, 0, 0, 0, 0, 1},
  {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 0, 0, 1, 1, 1}},
```

(b) 图像数据



(c) 处理后图像

```
#define XSIZE 8
#define YSIZE 8
for(j=0; j<YSIZE; j++)
    for(i=0; i<XSIZE; i++)
        *(image_out + j*XSIZE + i) =
        *(image_in + (YSIZE-1-j)*XSIZE + i);
```

(d) 处理程序

图 2-4

2.3 彩色图像

众所周知，各种彩色都是由 R、G、B 三个单色调配而成，各种单色都人为地从 0～255 分成了 256 个级，所以根据 R、G、B 的不同组合可以表示 256×256×256 = 16 777 216 种颜色，这种情况被称为全彩色图像（full−color image）或者真彩色图像（true−color image）。如果一幅图的每一个像素都用其 *R*、*G*、*B* 分量来表示，那么图像文件将会非常大。例如，一幅 640×480 像素的彩色图像，因为一个像素要用三个字节（一个字节是 8 位，总共 24 位）来表示 *R*、*G*、*B* 分量，所以需要保存 640×480×3=921 600 个字节。

对于一幅 640×480 像素的 16 色图像，如果每一个像素也用 *R*、*G*、*B* 三个分量表示，一个像素也要三个字节，这样保存整个图像也要约 1MB 的空间。但是 16 色图像，最多只表示 16 种颜色，上述保存方法因为有许多重复数据而浪费了空间。

为了减少保存的字节数，一般采用调色板（palette，或称 look up table，颜色表 LUT）技术来保存上述图像。如果做个颜色表，表中的每一行记录一种颜色的 R、G、B 值，即（*R*,*G*,*B*）。例如，第一行表示红色（255,0,0），那么当某个像素为红色时，只需标明索引 0 即可，这样就可以通过颜色索引来减少表示图像的字节数。对上述图像，如果用颜色索引的方法，我们来计算一下字节数，16 种状态，可以用 4 位（2^4）也就是半个字节来表示，整个图像数据需要用



640×480×0.5=153 600 个字节，另加一个颜色表的字节数。颜色表在 Windows 上是固定的结构格式，有四个参数，各占一个字节，前三个参数分别代表 R、G、B，第四个参数为备用，这样 16 个颜色的颜色表共需要 4×16=64 个字节。所以采用调色板技术表示上述 16 色图像时，总共需要 153 600+64=153 664 个字节，只占第一种保存方法的 1/6 左右，节省了许多存储空间。

而对于全彩色图像，直接用 R、G、B 分量来表示，使用调色板技术，相当于用前述保存方法保存一幅图像，并且调色板中也保存了一份同样的图像。

上述用 R、G、B 三原色表示的图像被称为位图（bitmap），有压缩和非压缩两种格式，后缀是 BMP。除了位图以外，图像的格式还有许多。例如，TIFF 图像，一般用于卫星图像的压缩格式，压缩时数据不失真，JPEG 图像，是被数码相机等广泛采用的压缩格式，压缩时有部分信号失真。由于在进行图像处理时，需要对非压缩的原数据进行处理，所以本书只介绍 BMP 的非压缩格式图像。

2.4 灰度图像

灰度图像是指只含亮度信息，不含色彩信息的图像。在 BMP 格式中没有灰度图像的概念，但是如果每个像素的 R、G、B 完全相同，也就是 $R=G=B=Y$，该图像就是灰度图像（或称 monochrome image，单色图像）。其中 Y 被称为灰度值，它应位于某个范围之内

$$Y_{min} \leqslant Y \leqslant Y_{max} \tag{2.1}$$

理论上要求 Y 仅为正的，且为有限值，区间$[Y_{min}, Y_{max}]$称为灰度级（gray scale 或 gray level），一般常用灰度级为[0,255]，这里 Y_{min} =0 为黑，Y_{max} =255 为白，所有中间值是从黑到白的各种灰色调，总共 256 级。

彩色图像可以由公式（2.2）变为灰度图像：

$$Y = 0.299R + 0.587G + 0.114B \tag{2.2}$$

2.5 位图文件格式

位图文件（bitmap file）的保存顺序如下：位图文件头 BITMAPFILEHEADER→位图信息头 BITMAPINFOHEADER→调色板 RGBQUAD（真彩色没有调色板）→图像数据。

1. 位图文件头BITMAPFILEHEADER

定义如下：



```
typedef struct tagBITMAPFILEHEADER {
        WORD      bfType;
        DWORD  bfSize;
        WORD   bfReserved1;
        WORD      bfReserved2;
        DWORD     bfOffBits;
} BITMAPFILEHEADER;
```

- bfType 文件类型，必须是0x424D，即字符串"MB"，代表位图文件。
- bfSize 指定文件大小，包括该结构体的14个字节。其中WORD为无符号16位整数，2个字节，DWORD为无符号32位整数，4个字节。
- bfReserved1，bfReserved2 保留，不用考虑。
- bfOffBits 指从文件头到实际位图数据的偏移字节数，即文件头、信息头和调色板的字节之和。

2．位图信息头BITMAPINFOHEADER

定义如下：

```
typedef struct tagBITMAPINFOHEADER{
        DWORD  biSize;
        LONG     biWidth;
        LONG     biHeight;
        WORD     biPlanes;
        WORD     biBitCount;
        DWORD  biCompression;
        DWORD  biSizeImage;
        LONG     biXPelsPerMeter;
        LONG     biYPelsPerMeter;
        DWORD    biClrUsed;
        DWORD    biClrImportant;
} BITMAPINFOHEADER;
```

- biSize 指该结构体的大小，DWORD为40个字节。LONG型是4个字节。
- biWidth 指图像的宽度，单位是像素。
- biHeight 指图像的高度，单位是像素。
- biplanes 必须是1。
- biBitCount 指图像数据位数。常用值有1（2^1色）、4（2^4色）、8（2^8色）、24（2^{24}



色）、32（2^{32}色）。

- biCompression　指定位图是否压缩。如果该值等于 BI_RGB，表示图像为非压缩格式。本书只讨论该种格式。
- biSizeImage　指图像数据的大小。biSizeImage = biWidth' × biHeight，biWidth'为大于或等于 biWidth 的最接近 4 的整倍数。如果 biCompression 为 BI_RGB，则该项可能为零。
- biXPelsPerMeter　指目标设备的水平分辨率，单位是每米的像素个数。
- biYPelsPerMeter　指目标设备的垂直分辨率，单位是每米的像素个数。
- biClrUsed　指图像用到的颜色数，如果该数为 0，则用到的颜色数为 2 的 biBitCount 次方。
- biClrImportant　指图像中重要的颜色数，如果该值为 0，则认为所有的颜色都是重要的。

3．调色板RGBQUAD

定义如下：

```
typedef struct tagRGBQUAD {
        BYTE    rgbBlue;
        BYTE    rgbGreen;
        BYTE    rgbRed;
        BYTE    rgbReserved;
} RGBQUAD;
```

- rgbBlue　该颜色的蓝色分量。
- rgbGreen　该颜色的绿色分量。
- rgbRed　该颜色的红色分量。
- rgbReserved　保留值。

4．图像数据

对于用到调色板的位图，图像数据就是该像素颜色在调色板中的索引值。对于真彩色图像，图像数据就是实际的 *R*、*G*、*B* 值，一个像素是由 3 个字节 24 位组成，第 1 个字节（前 8 位）表示 *B*，第 2 个字节（中间 8 位）表示 *G*，第 3 个字节（后 8 位）表示 *R*。

需要注意的是，BMP 文件是从下到上、从左到右排列的。即读文件时，最先读到的是图像最下面一行的左边第一个像素，最后读到的是最上面一行的最右一个像素。



2.6 采样与量化

如何把数字图像输入到计算机中呢？为了从如照片之类的模拟图像得到数字图像，必须通过采样和量化两个操作过程。

采样（sampling）是把空间上的连续的图像分割成离散的像素的集合。如图 2-5 所示，其中（a）是 512×512 像素，（b）是 256×256 像素，（c）是 128×128 像素，（d）是 64×64 像素，（e）是 32×32 像素，（f）是 16×16 像素。采样越细，像素越小，越能精细地表现图像。采样的精度有许多不同的设定，例如采用水平 256×垂直 256 像素、水平 512×垂直 512 像素、水平 640×垂直 480 像素等。

(a)(b)(c)
(d)(e)(f)



图 2-5

量化（quantization）是把像素的灰度（浓淡）变换成离散的整数值的操作。最简单的是用白（0）和黑（1）两个数值即 1 位（2 级）来量化，称为二值图像（binary image）。图 2-6 表示了量化位数与图像质量的关系，其中（a）是 8 bit（256 级），（b）是 6 bit（64 级），（c）是 4 bit（16 级），（d）是 3 bit（8 级），（e）是 2 bit（4 级），（f）是 1 bit（2 级）。量化越细致（位数越大），灰度级数（浓淡层次）表现越丰富，对于 6 位（64 级）以上的图像，几乎看不出有什么区别。考虑到在计算机内操作的方便性，一般采用 8 位（256 级），这意味着表示像素的灰度（浓淡）是 0～255 的数值。





图 2-6

对于彩色图像，需要对每个彩色成分 R、G、B 分别进行采样和量化，由于每种颜色 8 位，所以能够处理 $2^8\times2^8\times2^8$=16 777 216 色的图像。

2.7 图像处理的基本步骤

在使用图像处理方法时，只用一种方法就能解决问题的情况很少，大多是几种方法组合起来一起使用。例如在提取特定区域时，一般可以按照图 2-7 所示的基本步骤，顺序地进行图像处理。仅组合这些处理能够提取出物体来，不过这只对没有噪声、质量相当好的图像适用，通常的图像或多或少都有噪声混入，这时有必要增加消除噪声、灰度变换等处理使图像易于观看。另外，各个处理的参数也有必要进行多次校正，并观察相应的结果，最后根据结果选择适当的参数值。

可见，图像处理系统与图像的表示方式结合起来测试各个处理方法，根据输出图像，变更参数值，改变方法。对于上述处理我们将在后续章节中详细说明。



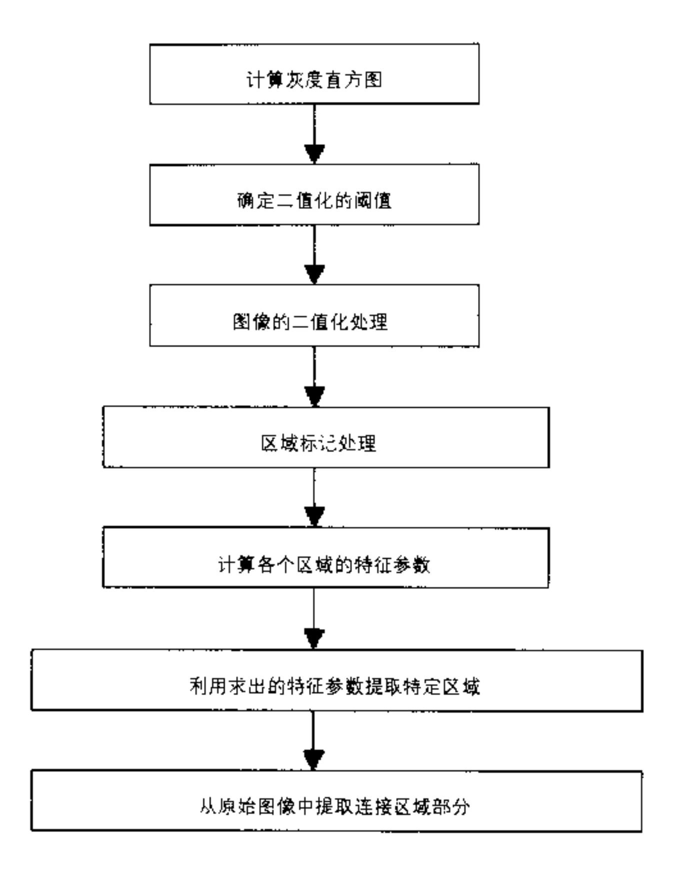


图 2-7

2.8 配套软件的使用方法

2.8.1 配套软件的介绍

本书提供专业版的源程序 Imgc 和学习版的源程序 Imgcx。专业版 Imgc 是一套完全可执行的 Visual C++界面源代码，界面内容除了图像的表示、读入、保存、彩色图像变灰度图像以外，还包含本书的所有内容，这些内容都附有图像处理函数的源代码。专业版 Imgc 主要面向教师和科研人员，适用于图像处理的课堂教学和科研。教师可以用专业版 Imgc 边向学生传授理论，边讲解处理程序，然后示范处理结果，使学生在实践中学习，可以完全改变传统理论灌输的旧教学模式。在科研上，研究者可以通过复制和改写图像处理的源程序，来达到自己的处理目的，



这样可以节省大量的编程时间。专业版 Imgc 对图像处理编程的自学者也非常有用，学习图像处理编程，最好的办法是对现有程序进行修改，然后查看处理结果的变化，该软件系统给自学者提供了在实践中学习的平台。

学习版 Imgcx 是一套完全可执行的界面源代码，界面内容包括图像的表示、读入和保存，主要用于课后编程练习和自学。对于大多数人来说，图像的表示、读入、保存等是学习图像处理编程的一大障碍。该软件给读者提供了图像表示、读入、保存的框架，排除了学习图像处理编程的障碍，可使读者轻松地进入图像编程状态。读者可以在 Imgcx 上添加菜单或对话框，将本书中各个独立的图像处理程序输入到 Imgcx 的框架中，起到在实际操作中轻松地学会图像处理编程的目的。

因为专业版 Imgc 是个成型的系统，如何利用和改造因人而异，所以本书不对其进行过多的说明，下面只介绍学习版 Imgcx 的使用方法。

2.8.2 安装学习版 Imgcx

本书附带的光盘里有 Imgcx 的安装程序，打开文件夹“学习版 Imgcx”执行 SetUp.exe，按提示安装即可。

安装后，用 Visual C++ 6.0 打开工程 Imgcx 即可进行编程。如果使用的是最新版本的 Visual C++ .NET，在第一次打开工程 Imgcx 时将提示项目升级，选择升级后，就自动将 Imgcx 升级到 Visual C++ .NET 版本，以后在 Visual C++ .NET 上即可编码执行。

2.8.3 图像处理函数的输入方法

下面以 Visual C++ 6.0 的界面为例说明图像处理函数的输入方法。

打开工程 Imgcx 以后，首先将自己需用的图像处理函数输入到工程 Imgcx 中。下面以一般二值化的函数为例进行说明。


操作步骤

① 打开 Imgcx 工程的菜单 File，选择 New 后，出现图 2–8 所示的 New 对话框。单击 New 对话框左上角的标签 Files。

② 如图 2–8 所示，在左侧的框内选择 C++ Source File，在右侧的 File 一栏输入函数名称 Threshold，在 Location 一栏一般会自动显示图像处理函数要放入的位置（本例中为 E:\Imgcx）。需要注意的是，要设置 Imgcx 在读者计算机系统上的实际安装位置，不一定是图 2–8 上的设置。Add to project 一栏是自动设定，如果不是图 2–8 的设定，应该按图 2–8 进行设定。完成上述设定以后，单击右下方的 OK 按钮，关闭 New 对话框。

③ 关闭 New 对话框以后，回到工程窗口。在工程窗口左侧的 FileView 中打开 Source Files，双击刚才新建的源程序文件 Threshold.cpp，在右侧的编辑框中会出现 Threshold.cpp 的空白编辑

区域。将 Threshold.cpp 的内容按第 3 章中的 "List 3.1" 如实输入，保存即可。参考图 2-9。

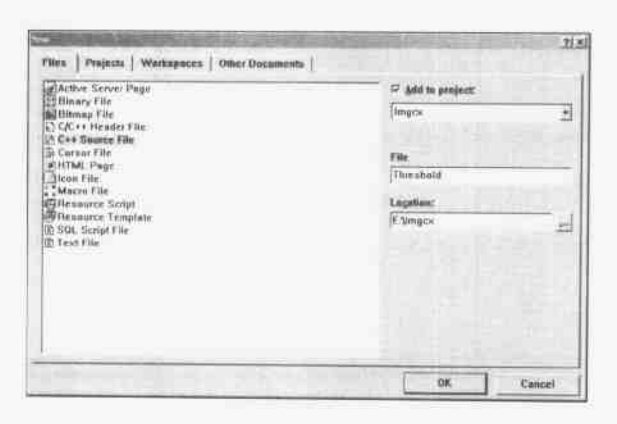


图 2-8

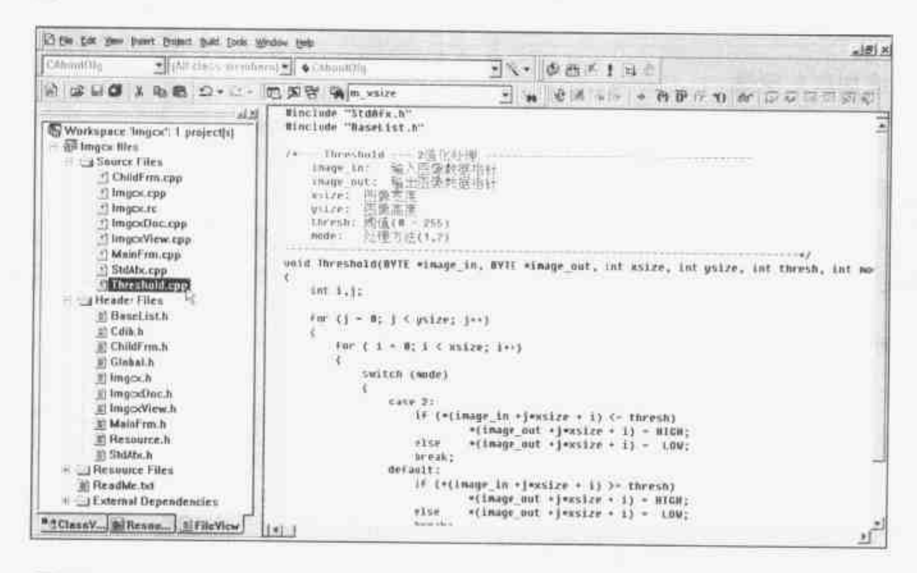


图 2-9

2.8.4 菜单的添加方法

1．打开菜单窗口

在工程窗口左侧的 ResourceView 中，打开 Menu，双击 IDR_IMGCXTYPE 后，右侧编辑对话区域出现 Imgcx 的菜单编辑窗口，如图 2-10 所示。





图 2-10

2．增加菜单目录

右击菜单后面的虚线方框，出现设置目录，如图 2-11 所示。

3．设置菜单名称

在设置目录里选择 Properties，出现设置对话框，在设置对话框里取消 Pop-up 的选择，然后设置菜单的 ID 和名称。图 2-12 是建立“2 值化处理”菜单的例子。设置后关闭设置对话框。



图 2-11



图 2-12

4．增设菜单函数

关闭设置对话框后，右击添加的菜单，在出现的设置目录上选择 ClassWizard（参考图 2-11），出现图 2-13 所示的 MFC ClassWizard 对话框。选择要增加菜单的 ID（如图 2-13 的 ID_BINARY）后，再选择 Messages 中的 COMMAND，然后单击右上侧的 Add Function 按钮，出



现设定函数名称的对话框。函数的名称是自动设定的，一般不需要改动。单击函数名称对话框的 OK 按钮后，函数增设完毕。

增设完函数以后，单击 ClassWizard 对话框右侧的 Edit Code 按钮即可进入函数编辑状态。菜单函数被增设在文件 ImgcxView.cpp 里。

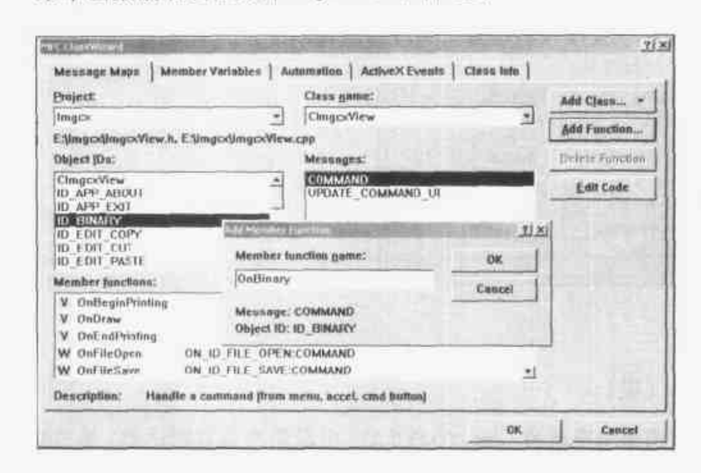


图 2-13

2.8.5 对话框的添加方法

1. 增设对话框

打开工程左侧的 ResourceView，右击 Dialog 目录后，在出现的菜单中选择 Insert Dialog（如图 2–14 所示）后，出现新加的对话框。



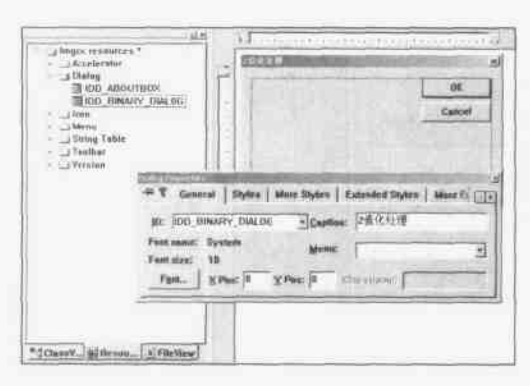


图 2-14　　*图 2-15*



2. 对话框的设定

右击新增的对话框，在出现的菜单中选择 Properties 后，弹出对话框的设置对话框，在设置对话框上单击标签 General，出现一般设置对话框，然后按图 2–15 所示设置对话框的 ID 和名称。再单击标签 More Styles，出现扩展设置对话框，选中 Visible，如图 2–16 所示。然后关闭设置对话框。对话框的大小以及按钮的位置，可以自由拖动设置。

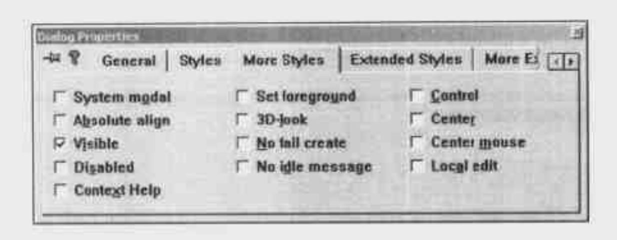


图 2-16

3. 建立对话框的Class（类）

右击对话框，在弹出的菜单中选择 ClassWizard 后，出现新类建立对话框，单击 OK 按钮后，出现 New Class 对话框。在新类建立对话框的 Name 一栏中输入新类的名称，例如 CBinaryDlg，如图 2–17 所示。


注意

类的名称前一般要加符号“C”，表示 Class，生成的文件名称 File name 将自动去掉该符号。以下各个栏目都是自动对应表示，一般不要变动。输入名称后，单击 OK 按钮关闭对话框，即完成新类的建立。

图 2-17

2.8.6 按钮、参数对话框的添加方法

1. 添加按钮

可以将对话框上自动生成的 OK 按钮和 Cancel 按钮改造后使用，也可以单击控制对话框的 Button（图 2–18 的箭头位置）后，单击对话框上的适当位置创建按钮。

在本例中，Cancel 作为关闭对话框的按钮不变，将 OK 按钮改为二值化的执行按钮。改造方法如下：右击原来的 OK 按钮，选择菜单中的 Properties 后，出现设定对话框，按图 2–19 所示设定即可。

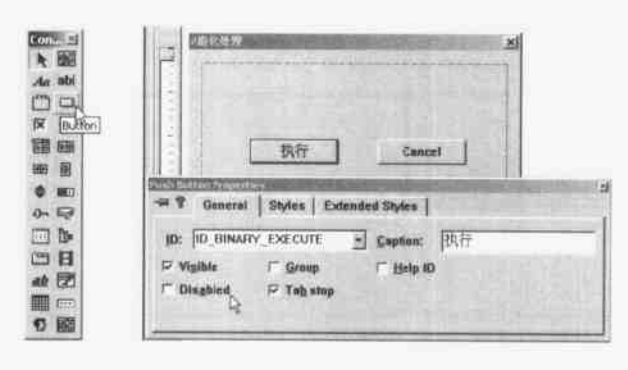


图 2-18　　*图 2-19*

2. 添加参数窗口及定义参数

1 / 静态文字

单击控制对话框的 Static Text（静态文字，图 2–18 上的 **Aa**）后，再单击对话框上的适当位置放置该按钮。放置后，右击该按钮，打开 Properties 对话框，设定名称为“阈值”。静态文字的 ID 默认是 ID_STATIC，最好不要改变，如图 2–20 所示。

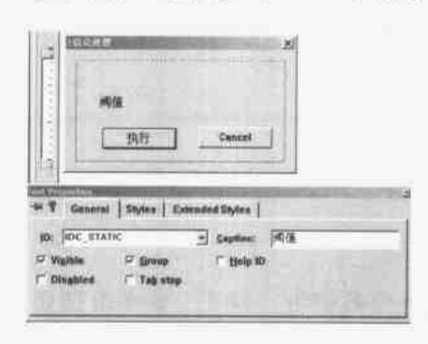


图 2-20

2 / 输入对话框

单击控制对话框的 Edit Box（编辑对话框，图 2-18 上的 **ab|**）后，再单击对话框上的“阈值”右侧放置该按钮。放置后，右击该按钮，打开 Edit Properties 对话框，设定 ID 为 IDC_THRESHOLD_EDIT，如图 2-21 所示。

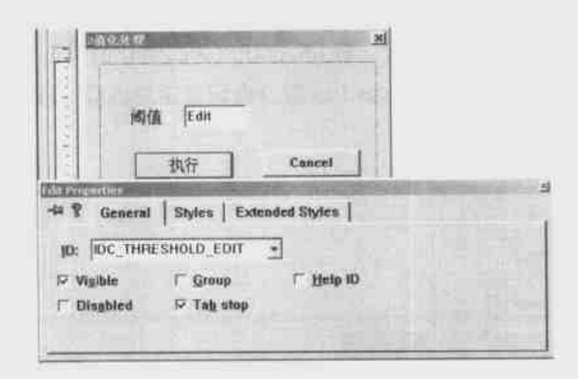


图 2-21

3 / 单选按钮

单击控制对话框的 Radio Button（单选按钮，图 2-18 上的 ◉）后，再单击对话框上的适当位置放置该按钮。放置后，右击该按钮，打开 Edit Properties 对话框，按图 2-22 进行设定。注意选中 Group 复选框。

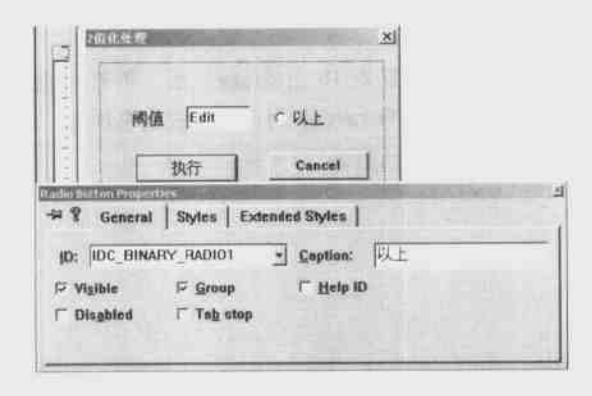


图 2-22

用同样的方法在该按钮的右侧再建立一个名字为“以下”的单选按钮，ID 设为 IDC_BINARY_RADIO2，如图 2-23 所示。注意取消 Group 复选框的选中状态。



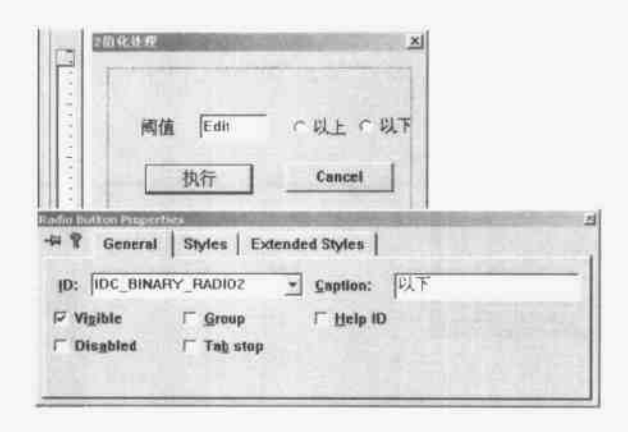


图 2-23

4 / 定义参数

右击编辑对话框 IDC_THRESHOLD_EDIT，在弹出的快捷菜单里选择 ClassWizard 后，弹出 MFC ClassWizard 对话框。在 ClassWizard 对话框中单击标签 Member Variables。在 Control IDs 下面选择对象 IDC_THRESHOLD_EDIT 之后，单击右侧的 Add Variable 按钮，出现参数设定对话框。如图 2-24 所示，对 IDC_THRESHOLD_EDIT 的参数进行设定。设定后单击 OK 按钮关闭参数设置对话框，单击 OK 按钮关闭 ClassWizard 对话框。

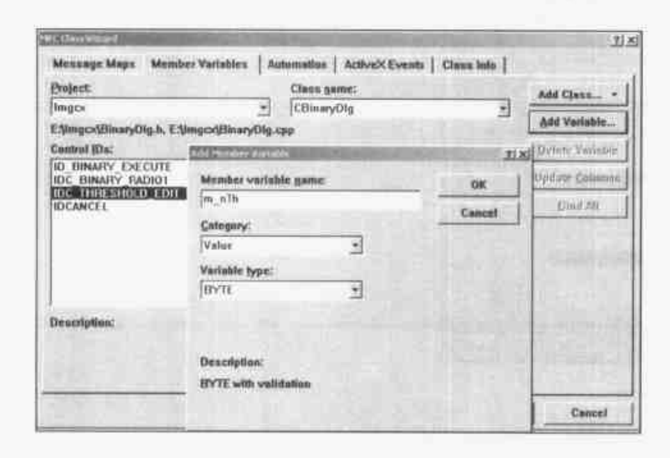


图 2-24

以同样步骤，给单选按钮“以上”定义参数，名字为 m_nModel，值型（Variable type）为 int，如图 2-25 所示。



图 2-25

2.8.7　函数的添加方法

1．添加对话框的初始化函数

初始化函数在对话框打开时自动执行，可以在初始化函数里对参数等进行初始设定。

右击对话框，打开 MFC ClassWizard 对话框，如图 2–26 所示，在 Class Name 和 Object IDs 两个框中都选择 CBinaryDlg，在 Messages 框中选择 WM_INITDIALOG，再单击对话框右侧的 Add Function 按钮，然后，单击对话框下方的 OK 按钮关闭对话框，或者单击 Edit Code 进入函数编辑对话框。

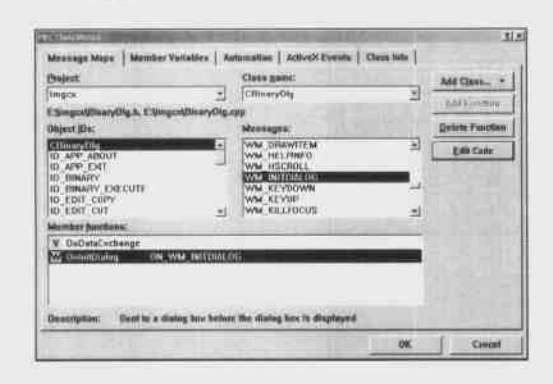


图 2-26

2．添加命令函数

右击对话框上的"执行"按钮，在弹出的快捷菜单中选择 ClassWizard，弹出 WFC ClassWizard 对话框，单击标签 Message Maps，在 Object IDs 一栏中选择对应的 ID，在 Messages 一栏中选择 BN_CLICKED（单击执行），然后单击对话框右侧的 Add Function 按钮，弹出函数添加对话



框（图 2–27 中间对话框）。在函数对话框内自动出现函数名称，一般不要改动该名称。单击 OK 按钮关闭函数对话框，再单击 OK 按钮关闭 WFC ClassWizard 对话框。以同样方法也给 Cancel 按钮添加个函数。

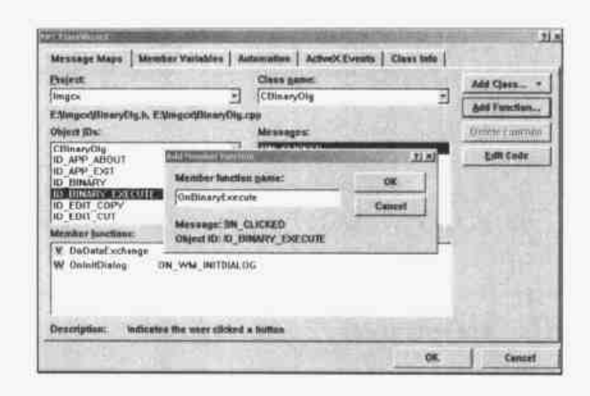


图 2-27

2.8.8　将对话框与菜单连接

要想打开程序，单击菜单后执行上述完成的对话框，必须将对话框与相应的菜单进行连接。下面把对话框“二值化处理”与菜单“二值化处理”连接起来。在前面介绍的菜单的添加方法里，已经介绍菜单函数添加在文件 ImgcxView.cpp 里的方法，所以必须将对话框与该文件里的菜单函数进行连接。连接方法如下所示。



操作步骤

1 在源程序文件 ImgcxView.cpp 的上方加入对话框的头文件 BinaryDlg.h。

2 在源程序文件 ImgcxView.cpp 中的 OnBinary( )函数里加入调用对话框的内容。

具体增加内容如下列 List 2.1 的源程序文件所示。

List 2.1　源程序ImgcxView.cpp

```
// ImgcxView.cpp : implementation of the CImgcxView class
//
#include "stdafx.h"
#include "Imgcx.h"

#include "ImgcxDoc.h"
#include "ImgcxView.h"
#include "Cdib.h"
#include "Global.h"
```



```
#include "BinaryDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CImgcxView

IMPLEMENT_DYNCREATE(CImgcxView, CScrollView)

BEGIN_MESSAGE_MAP(CImgcxView, CScrollView)
   //{{AFX_MSG_MAP(CImgcxView)
   ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
   ON_COMMAND(ID_FILE_SAVE, OnFileSave)
   ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
   ON_COMMAND(ID_BINARY, OnBinary)
   //}}AFX_MSG_MAP
   // Standard printing commands
   ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CImgcxView construction/destruction

CImgcxView::CImgcxView()
{
   // TODO: add construction code here
   m_nXSize = 640; //初始化图像窗口的宽度
   m_nYSize = 480; //初始化图像窗口的高度

   //初始化图像数据指针
```



```
    m_pImage = NULL;
    m_pImageR = NULL;
    m_pImageG = NULL;
    m_pImageB = NULL;

    m_bFileOpen = FALSE;

}

CImgcxView::~CImgcxView()
{
    //解散图像数据内存
    if(m_pImage != NULL)
    {
        delete m_pImage;
        m_pImage = NULL;
    }
    if(m_pImageR != NULL)
    {
      delete m_pImageR;
      m_pImageR = NULL;
    }
    if(m_pImageG != NULL)
    {
      delete m_pImageG;
      m_pImageG = NULL;
    }
    if(m_pImageB != NULL)
    {
      delete m_pImageB;
      m_pImageB = NULL;
    }

    ::DeleteDispDib();
}

BOOL CImgcxView::PreCreateWindow(CREATESTRUCT& cs)
```



```
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CImgcxView drawing

void CImgcxView::OnDraw(CDC* pDC)
{
    CImgcxDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CDib *pDib ;
    HGDIOBJ hOrg ;
    CPalette *pOrgPal ;

    ///////////////////////////////////////////////
    // Get current image
    pDib = ::GetDib() ;

    /////
    hOrg = ::SelectObject(m_cMemoryDC.m_hDC, m_hBitmap) ;


      // Draw image
    pDib->UsePalette(&m_cMemoryDC) ;
    pDib->Draw(&m_cMemoryDC, CPoint(0, 0), CSize(GetXSize(), GetYSize())) ;

    // set paletter
    pOrgPal=(CPalette*)pDC->SelectPalette(pDC->GetCurrentPalette(),FALSE);
    // view
    pDC->BitBlt(0, 0, GetXSize(), GetYSize(), &m_cMemoryDC, 0, 0, SRCCOPY);
    // recover paletter
    pDC->SelectPalette(pOrgPal, FALSE) ;
```



```
    ////////////////////////////////////////////
    // release DIB
    ::SelectObject(m_cMemoryDC.m_hDC, hOrg) ;
}

void CImgcxView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    //设定滚动轴的大小
    SetWindowSize(m_nXSize, m_nYSize);

      /////// initialize
    CClientDC dc(this);

    ::CreateDispDib(&dc, m_nXSize, m_nYSize);

    CDib *pDib ;
    pDib = ::GetDib() ;

    //表示当前图像
    // made DIB bitmap
    if ((m_hBitmap = ::CreateDIBitmap(dc.m_hDC, pDib->m_lpBMIH, 0L, NULL, NULL,
    0)) != 0) {
      // initialize structure data by GetDIBits()
      ::GetDIBits(dc.m_hDC, m_hBitmap, 0, m_nYSize, NULL, (LPBITMAPINFO)(pDib->
      m_lpBMIH), DIB_RGB_COLORS);
      // made memory DC
      m_cMemoryDC.CreateCompatibleDC(&dc) ;
    }
    ////////////////////////////////////////////

    Invalidate();
}

////////////////////////////////////
```



```
// CImgcxView printing

BOOL CImgcxView::OnPreparePrinting(CPrintInfo* pInfo)
{
     // default preparation
     return DoPreparePrinting(pInfo);
}

void CImgcxView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
     // TODO: add extra initialization before printing
}

void CImgcxView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
     // TODO: add cleanup after printing
}

////////////////////////////////////////////
// CImgcxView diagnostics

#ifdef _DEBUG
void CImgcxView::AssertValid() const
{
     CScrollView::AssertValid();
}

void CImgcxView::Dump(CDumpContext& dc) const
{
     CScrollView::Dump(dc);
}

CImgcxDoc* CImgcxView::GetDocument() // non-debug version is inline
{
     ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CImgcxDoc)));
     return (CImgcxDoc*)m_pDocument;
}
```



```
#endif //_DEBUG

/////////////////////////////////////////////
// CImgcxView message handlers

void CImgcxView::OnFileOpen()
{
     //读入位图文件
     ::Load_imagefile_bmp();

     m_nXSize = ::GetXSize();
     m_nYSize = ::GetYSize();
       //设定图像窗口大小
     SetWindowSize(m_nXSize, m_nYSize);
     //更新画面
     Invalidate();
     //判断图像格式
     if( ::GetImageType() == 24)
     {
       //消除原来图像数据
     if(m_pImageR != NULL)
     {
     delete[] m_pImageR;
     m_pImageR = NULL;
     }
     if(m_pImageG != NULL)
     {
     delete[] m_pImageG;
     m_pImageG = NULL;
     }
     if(m_pImageB != NULL)
     {
     delete[] m_pImageB;
     m_pImageB = NULL;
     }

     //为新图像分配内存
```



```
	m_pImageR = new BYTE[m_nXSize*m_nYSize];
	m_pImageG = new BYTE[m_nXSize*m_nYSize];
	m_pImageB = new BYTE[m_nXSize*m_nYSize];
	//读入新图像数据
	::ReadImageDataRGB(m_pImageR, m_pImageG, m_pImageB);
	}
	else if(::GetImageType() == 8)
	{
	//消除原来图像数据
	if(m_pImage != NULL)
	{
	delete[] m_pImage;
	m_pImage = NULL;
	}

	//为新图像分配内存
	m_pImage = new BYTE[m_nXSize*m_nYSize];
	//读入新图像数据
	::ReadImageData(m_pImage);
	}

	m_bFileOpen = TRUE;

	//更新画面
	Invalidate();
}

void CImgcxView::OnFileSave()
{
	if(!m_bFileOpen)//没有读入图像
	return;
	::Save_imagefile_bmp();
}

void CImgcxView::OnFileSaveAs()
{
	if(!m_bFileOpen)//没有读入图像
```



```
        return;

        ::SaveAs_imagefile_bmp();
    }

    void CImgcxView::SetWindowSize(int xsize, int ysize)
    {
        //设定滚动轴的大小
        SetScrollSizes(MM_TEXT, CSize(xsize-1, ysize-1));

        /////////初始化表示帧的大小
        CRect cRect ;
        CMDIChildWnd* pChildFrm = (CMDIChildWnd*)GetParentFrame();
        // 改变表示的大小
        cRect.SetRect(0, 0, xsize, ysize) ;
        CalcWindowRect((LPRECT)cRect, CWnd::adjustOutside) ;
        SetWindowPos(&wndTopMost,cRect.left,cRect.top,cRect.Width(),cRect.Height(),
        SWP_NOZORDER|SWP_NOMOVE) ;

        // 改变帧的大小
        pChildFrm->CalcWindowRect((LPRECT)cRect) ;
        pChildFrm->SetWindowPos(NULL, cRect.left, cRect.top, cRect.Width(), cRect.
        Height(), SWP_NOZORDER|SWP_NOMOVE) ;
        ///////////////////////////////////////////////
        }
        void CImgcxView::OnBinary()
        {
        //定义 dlg
        CBinaryDlg dlg(this);
        //执行对话框
        dlg.DoModal();
    }
```

加完上述内容后，对工程进行编码，然后执行命令，在窗口上单击菜单“二值化处理”即可表示对话框。

上述对话框是模式对话框，所谓模式对话框是指在关闭该对话框之前，对话框以外的命令



不能执行。与模式对话框相对的是非模式对话框，由于篇幅所限，请读者参考其他 Visual C++ 的参考书。

2.8.9 作成对话框函数

下列 List 2.2 表示完成的头文件 BinaryDlg.h，List 2.3 表示完成的源程序 BinaryDlg.cpp，黑体字是另加的内容，在自己的函数里加上黑体字部分即可。加黑体字部分后，对工程编码，即可执行。

List 2.2 头文件BinaryDlg.h

```
#if!defined(AFX_BINARYDLG_H__2CDE14C6_A8F3_44C6_9C11_E36072348405__INCLUDED_)
#define AFX_BINARYDLG_H__2CDE14C6_A8F3_44C6_9C11_E36072348405__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// BinaryDlg.h : header file
//

/////////////////////////////////////////
// CBinaryDlg dialog

class CBinaryDlg : public CDialog
{
// Construction
public:
   CBinaryDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
   //{{AFX_DATA(CBinaryDlg)
   enum { IDD = IDD_BINARY_DIALOG };
   BYTE  m_nTh;
   int   m_nModel;
   //}}AFX_DATA

   //定义主窗口指针
   CWnd *m_pParent;
```



```
    //定义图像指针
    BYTE *m_pImage_in;
    BYTE *m_pImage_out;

    //定义图像大小
    int m_nxsize;
    int m_nysize;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBinaryDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CBinaryDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnBinaryExecute();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
    the previous line.

#endif
//!defined(AFX_BINARYDLG_H__2CDE14C6 A8F3_44C6_9C11_E36072348405__INCLUDED_)
```

List 2.3 源程序BinaryDlg.cpp

```
// BinaryDlg.cpp : implementation file
//
```



```
#include "stdafx.h"
#include "Imgcx.h"
#include "BinaryDlg.h"

//包含有图像表示、存取等函数的头文件
#include "Global.h"

//包含图像处理函数的头文件
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// CBinaryDlg dialog

CBinaryDlg::CBinaryDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CBinaryDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CBinaryDlg)
   m_nTh = 100;
   m_nModel = 0;
   //}}AFX_DATA_INIT

   //保存主窗口句柄
   m_pParent = pParent;

   //初始化图像指针
   m_pImage_in = NULL;
   m_pImage_out = NULL;
}

void CBinaryDlg::DoDataExchange(CDataExchange* pDX)
{
```



```
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBinaryDlg)
	DDX_Text(pDX, IDC_BINARY_EDIT, m_nTh);
	DDX_Radio(pDX, IDC_BINARY_RADIO1, m_nModel);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CBinaryDlg, CDialog)
	//{{AFX_MSG_MAP(CBinaryDlg)
	ON_BN_CLICKED(ID_BINARY_EXECUTE, OnBinaryExecute)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////
// CBinaryDlg message handlers

BOOL CBinaryDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //获得图像大小
    m_nxsize = ::GetXSize();
    m_nysize = ::GetYSize();

//彩色图像时退出
 if(::GetImageType() == 24)
   return FALSE;

 //分配内存
 m_pImage_in = new BYTE[m_nxsize*m_nysize];
 m_pImage_out = new BYTE[m_nxsize*m_nysize];

 return TRUE;  // return TRUE unless you set the focus to a control
             // EXCEPTION: OCX Property Pages should return FALSE
}
```



```
void CBinaryDlg::OnBinaryExecute()
{
     // TODO: Add your control notification handler code here

     //获得窗口数据
     UpdateData(TRUE);

     //读入图像数据
     ::ReadImageData(m_pImage_in);

     //二值化处理
     Threshold(m_pImage_in,m_pImage_out,m_nxsize,m_nysize,m_nTh,m_nModel+1);

     //表示处理结果
     ::Disp_image(m_pImage_out);

     //更新图像画面
     m_pParent->Invalidate();

}

void CBinaryDlg::OnCancel()
{
     // TODO: Add extra cleanup here

     //释放内存
     delete [] m_pImage_in;
     delete [] m_pImage_out;

     CDialog::OnCancel();
}
```

2.9 配套函数的说明

2.9.1 图像处理源程序

List 2.4 列出了本书中图像处理源程序的一览表，各个函数中引数的含义可以参看各章后



面该函数前面的说明。读者可以在学习了各章节的内容之后，参考 2.8 节的例子，将各章后面的图像处理函数逐个加入到 lmgcx 系统里。

List 2.4　图像处理源程序一览表

```
// BaseList.h

#ifndef  __IMGCHEADERFILE__
#define  __IMAGCHEADERFILE__

/*-----------------定义参数-----------------------*/
//函数返回值
#define   OK  0
#define   NG -1

// 白像素
#define HIGH   255

//黑像素
#define LOW      0

//连接成分的基数值
#define  L_BASE  100

//直方图图像的基数值
#define BIAS 128

//π值
#define  PI    (float)3.14159265

//√2值
#define  ROOT2 (float)1.41421356

//判断有无饱和度的阈值
#define  THRESHOLD 0.0

//无饱和度时的代入值
#define  NONE    0.0
```



```
//图像区域以外时的像素值
#define  B_VAL  128

//一个字节的位数
#define  BYTESIZE  8

//可变单位（2 或 4 位）
#define  LEN     4

//像素位置(x,y)及周边像素的亮度和(weight)
struct XYW {
   int  x, y, w;
};

/*------------------------区域分割与提取---------------------------*/

//一般二值化处理
void Threshold(BYTE *image_in, BYTE *image_out, int xsize, int ysize, int thresh,
int mode);

// 双阈值二值化处理
void Threshold_mid(BYTE *image_in, BYTE *image_out, int xsize, int
ysize, int thresh_low, int thresh_high);

//反转图像
void Reverse_image(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

//像素分布直方图
void Histgram(BYTE *image, int xsize, int ysize, long hist[256]);

//计算直方图百分比
void CalHistPercent(long hist[], float hist_radio[], float &max_percent);

//直方图平滑化
void Hist_smooth(long hist_in[256], long hist_out[256]);
```



```
//直方图图像化（图像宽度大于等于64）
void Hist_to_image(long hist[256], BYTE* image_hist, int xsize, int ysize);

/*-------------------------边缘检测与提取----------------------------*/

//1阶微分边缘检测（梯度算子）
void Differential(BYTE*image_in,BYTE*image_out, int xsize, int ysize, float amp);

//2阶微分边缘检测（拉普拉斯算子）
void Differential2(BYTE *image_in,BYTE*image_out,int xsize,int ysize,float amp);

//Prewitt法边缘检测
void Prewitt(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float amp);

//二值图像的细线化处理
void Thinning(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

/*--------------------------图像平滑-----------------------------*/

//消除噪声处理（移动平均）
void Image_smooth(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

//消除噪声处理（中值）
void Median(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

//膨胀
void Dilation(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

//腐蚀
void Erodible(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

/*--------------------------图像增强-----------------------*/

//亮度n倍
void Brightness_amplify(BYTE *image_in, BYTE *image_out, int xsize, int ysize,
float n);
```



```
//求亮度范围
void Brightness_range(BYTE *image_in, int xsize, int ysize, int *fmax, int *fmin);

//亮度范围扩张
void Brightness_expand(BYTE *image_in, BYTE *image_out, int xsize, int ysize,
int fmax, int fmin);

//直方图均衡化
void Hist_plane(BYTE *image_in, BYTE *image_out, int xsize, int ysize, long
hist[256]);
/*-------------------------特征选择与描述----------------------------*/

//区域标记
int Labeling(BYTE *image_in, BYTE *image_out, int xsize, int ysize, int *cnt);

//测量特征参数
void Features(BYTE*image_label_in,BYTE*image_label_out,int xsize,int ysize,int
cnt,float size[],float length[],float ratio[],int center_x[],int center_y[]);

//提取具有某圆形度的对象物
void Ratic_extract(BYTE*image_label_in,BYTE*image_label_out,int xsize,int ysize,
 int cnt, float ratio[], float ratio_min, float ratio_max);

//提取某面积范围的对象物
void Size_extract(BYTE *image_label_in, BYTE *image_label_out, int xsize, int
ysize,int cnt, float size[], float size_min, float size_max);

//复制模块领域的原图像
void Mask_copy(BYTE*image_in,BYTE*image_out,BYTE*image_mask,int xsize,int ysize);

/*--------------------------彩色变换-----------------------------*/

//做彩色条码
void Colorbar(BYTE *image_r, BYTE *image_g, BYTE *image_b,int xsize, int ysize,
int level);

//由R、G、B变换亮度、色差信号
```



```
void Rgb_to_yc(BYTE *image_r, BYTE *image_g, BYTE *image_b,
 int *y, int *c1, int *c2, int xsize, int ysize);

//色差信号图像化
void Yc_to_image(int *data, BYTE *image, int xsize, int ysize);

//由亮度、色差变换R、G、B信号
void Yc_to_rgb(int *y, int *c1, int *c2, BYTE *image_r, BYTE *image_g,
   BYTE *image_b, int xsize, int ysize);

//由色差信号计算饱和度和色调
void C_to_SH(int *c1, int *c2, int *sat, int *hue, int xsize, int ysize);

//由饱和度和色调计算色差信号
void SH_to_C(int *sat, int *hue, int *c1, int *c2, int xsize, int ysize);

//由饱和度数据变换灰度图像
int Sat_to_image(int *sat, BYTE *image_out, int xsize, int ysize);

//由某点的RGB值计算饱和度和色调
void Rgb_to_SH(BYTE r, BYTE g, BYTE b, double *sat, double *hue);

//由色调数据变换灰度图像
void Hue_to_image(int *sat, int *hue, double stdhue,BYTE *image_out, int xsize,
int ysize);

//改变亮度、饱和度和色调
void Change_YSH(int *in_y, int *in_sat, int *in_hue, int *out_y,
 int *out_sat, int *out_hue, float ym, float sm, float hd, int xsize, int ysize);

/*------------------------彩色分割----------------------*/

//计算二维直方图并图像化
void Hist2_image(BYTE *image_in1, BYTE *image_in2, BYTE *image_hist, int xsize,
int ysize);

//R、G、B的阈值处理
```



```
void Thresh_color(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
BYTE *image_out_r, BYTE *image_out_g, BYTE *image_out_b,int thdrl, int thdrm,
int thdgl, int thdgm, int thdbl, int thdbm,int xsize, int ysize);

//生成硬合成键
void Hard_key(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
 BYTE *image_key, int xsize, int ysize, int thresh);

//生成软合成键
void Soft_key(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
 BYTE *image_key, int xsize, int ysize, int thdh, int thdl);

// 图像硬合成
void Synth(BYTE *image_in1_r, BYTE *image_in1_g, BYTE *image_in1_b, BYTE
*image_in2_r,BYTE *image_in2_g, BYTE *image_in2_b, BYTE *image_out_r, BYTE
*image_out_g,BYTE *image_out_b, BYTE *image_key, int xsize, int ysize);

//图像合成（消除境界线）
void S_synth(BYTE *image_in1_r, BYTE *image_in1_g, BYTE *image_in1_b,
BYTE *image_in2_r, BYTE *image_in2_g, BYTE *image_in2_b,
BYTE *image_out_r, BYTE *image_out_g, BYTE *image_out_b,
BYTE *image_key, int xsize, int ysize);

/*--------------------------几何变换-----------------------------*/

//错误的放大缩小法
void Scale_NG(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float zx,
float zy);

//放大缩小（最近邻点法）
void Scale_near(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float zx,
float zy);

//放大缩小（双线性内插法）
void Scale(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float zx, float
zy);
```



```
//位移（双线性内插法）
void Shift(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float px, float
py);

//旋转（双线性内插法）
void Rotation(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float deg);

//仿射变换（移动、旋转、放大缩小）
void Affine(BYTE*image_in,BYTE *image_out, int xsize, int ysize, float deg,
float zx, float zy, float px, float py);

//透视变换（双线性内插法）
void Perspective(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float ax,
float ay,
float px, float py, float pz, float rz,
float rx, float ry, float v, float s);

/*-----------------傅立叶变换---------------------------------*/

//1 次傅立叶变换
int FFT1(float a_rl[], float a_im[], int ex, int inv);

//1 次傅立叶变换的主计算部分
void fft1core(float a_rl[], float a_im[], int length,
 int ex, float sin_tbl[], float cos_tbl[], float buf[]);

//2 次傅立叶变换
int FFT2 (float *a_rl, float *a_im, int inv, int xsize, int ysize);

//将 2 次 FFT 的变换结果频率域图像化
int FFTImage(BYTE *image_in, BYTE *image_out, int xsize, int ysize);

//2 次 FFT 的滤波处理、滤波后的频率域图像化
int FFTFilter(BYTE*image_in,BYTE*image_out, int xsize, int ysize, int a, int b);

//图像的 2 次 FFT 变换、滤波处理、傅立叶逆变换
int FFTFilterImage(BYTE*image_in,BYTE*image_out,int xsize,int ysize,int a,int b);
```

```
/*---------------------图像压缩-------------------------------*/

//预测编码 DPCM （预测法(1)，处理一行区域）
void Dpcm1(BYTE* image_in, int xsize, int line, short *data_out);

//预测编码 DPCM （预测法(2)，处理一行区域）
void Dpcm2(BYTE* image_in, int xsize, int line, short *data_out);

//DPCM 数据分布直方图
void Histgram_dpcm(BYTE *image, int xsize, int ysize, long hist[512]);

//计算 DPCM 直方图百分比
void CalHistPercent_dpcm(long hist[],float hist_radio[],float &max_percent);

//DPCM 的解码 （预测法(1) 处理一行区域）
void Idpcm1(short *data_in, int xsize, int line, BYTE* image_out);

//DPCM 的解码 （预测法(2)：处理一行区域）
void Idpcm2(short *data_in, int xsize, int line, BYTE* image_out);

//变长编码
int Vlcode(short int data_in[], int no, char vlc_out[]);

//变长编码的解码
void Ivlcode(char vlc_in[], int no, short int data_out[]);

//由 DPCM 码到变长码的变换
int Event(short dt);

//由变长码到 DPCM 码的转换
int Ievent(short ev);

//DPCM + 变长编码
int Dpcm_vlcode(BYTE *image_in, int xsize, int ysize, BYTE* image_buf);

//DPCM + 变长编码的解码
int Idpcm_vlcode(BYTE *image_buf, BYTE *image_out, int xsize, int ysize);
```



```
/*-------------------小波变换------------------------------*/

//二维小波变换
void Wavelet2d (BYTE *image_in,int xsize,int ysize,double *s1, double *w1v,
double *w1h, double *w1d);

//二维小波信号图像化
void Wavelet2d_image (double *s1, double *w1v, double *w1h, double *w1d,
BYTE *image_out, int xsize, int ysize);

//二维小波逆变换
void Iwavelet2d (double *s1, double *w1v, double *w1h, double *w1d,
BYTE *image_out, int xsize, int ysize );

//一维小波变换
void Wavelet1d (double *s0, int s_len,  double *p, double *q,
int sup, double *s1, double *w1);

//一维小波逆变换
void Iwavelet1d (double *s1, double *w1, int s_len, double *p,
double *q, int sup, double *s0);
#endif
```

2.9.2 图像表示、存取函数

List 2.5 中列出了图像表示、存取等函数的一览表，这些函数是以库函数的方式提供的，在使用这些函数时，需要在调用这些函数的文件前面加上 #include "Global.h" （参考 List 2.3）。具体使用方法可以参考"附录"里各个 Visual C++界面的源程序。

List 2.5　图像表示、存取等DLL函数一览表

```
//Global.h
class CDib;

#ifdef  __EXPORT
#define  EXIMPORT extern "C" __declspec( dllexport)
#else
```



```
#define  EXIMPORT extern "C" __declspec( dllimport)
#endif

//建立图像表示用 Dib
// pDC 图像表示装置指针
// xsize 表示图像的宽度
// ysize 表示图像的高度
EXIMPORT int CreateDispDib(CDC* pDC, int xsize, int ysize);

//消除 Dib
EXIMPORT void DeleteDispDib( );

//建立参考窗口读入图像
EXIMPORT int Load_imagefile_bmp( );

//直接输入文件名读入图像
   // filename 读入文件的名称（包括路径）
EXIMPORT int Load_original_image(CString filename);

//读灰度图像数据到设定内存
   // image 存储图像数据的指针（要事先分配内存）
EXIMPORT int ReadImageData(BYTE *image);

//读彩色图像数据到设定内存
// imageR 存储图像 R 分量数据的指针（要事先分配内存）
// imageG 存储图像 G 分量数据的指针（要事先分配内存）
// imageB 存储图像 B 分量数据的指针（要事先分配内存）
EXIMPORT int ReadImageDataRGB(BYTE *imageR, BYTE *imageG, BYTE *imageB);

//图像保存
EXIMPORT BOOL Save_imagefile_bmp();

//图像另存为
EXIMPORT BOOL SaveAs_imagefile_bmp();

//获得图像横向大小
EXIMPORT int GetXSize();
```



```
//获得图像纵向大小
EXIMPORT int GetYSize();

//获得图像数据指针
EXIMPORT LPBYTE GetImage();

//获得图像类型 (8 = 灰度、24 = 彩色)
EXIMPORT int GetImageType();

//表示内存内的灰度图像
    // image 要表示的灰度图像数据指针
EXIMPORT void Disp_image(BYTE *image);

//表示内存内的彩色图像
// imageR 要表示的彩色图像的 R 分量数据指针
// imageG 要表示的彩色图像的 G 分量数据指针
// imageB 要表示的彩色图像的 B 分量数据指针
EXIMPORT void Disp_imageRGB( BYTE *imageR, BYTE *imageG, BYTE *imageB);

//获得表示图像的名称
    // cFileName 目前表示的图像的名字 (包括路径)
EXIMPORT void GetImageFileName(char *cFileName);

//设定表示图像的名称
    // fn 目前表示的图像的名字 (包括路径)
EXIMPORT void PutImageFileName(CString fn);

//获得表示的 Dib
EXIMPORT CDib* GetDib( );

//设定表示的 Dib
EXIMPORT void PutDib(CDib *pDib);

//彩色图像变成灰度图像
EXIMPORT int Color_to_mono();
```





第3章

区域的分割与提取

3.1　如何提取物体

有时要求从图像中除去不需要的东西，只提取出需要的东西。例如以下几种情况：

- 在街景中只提取人。
- 从邮件中查找邮政编码来进行分类。
- 判别苹果的大小，依据其大小进行分类。
- 使用监视摄像机，当发现有贸然进入的人时发送警报。
- 判断钱币的种类、真伪等。

不同的工作有不同的要求，如何满足这些要求呢？图 3–1 所示的图像中包含了很多内容，既有人物，也有猫、狗、鸟等动物，还有花草、树木、城堡等景物。如果从中只提取人物，要注意到人物的灰度、颜色、形状上的不同，可以使用各种各样的方法，但最简单的方法是阈值处理（threshold）。本章将具体讲解如何使用阈值处理方法。



图 3-1

3.2　基于阈值的区域分割与提取

如图 3–2 所示，要从图中提取出钳子就可以使用阈值处理的方法。阈值处理就是对于输入图像的各像素，灰度值在某定值（称为阈值，threshold）范围内时，赋予对应输出图像的像素

为白色（$HIGH=255$）或黑色（$LOW=0$）。阈值处理可用式（3.1）或式（3.2）表示。

$$g(x,y)=\begin{cases}HIGH & f(x,y)\geq t\\ LOW & f(x,y)<t\end{cases} \tag{3.1}$$

$$g(x,y)=\begin{cases}HIGH & f(x,y)\leq t\\ LOW & f(x,y)>t\end{cases} \tag{3.2}$$

其中 $f(x,y)$、$g(x,y)$ 分别是处理前和处理后的图像在（x,y）处像素的灰度值，t 是阈值。通过上述的阈值处理后，得到只有两个灰度值的二值图像（binary image），所以一般也被称作二值化处理（binarization）。处理程序表示在 List 3.1 中，其中当 mode=1 时选择式（3.1），当 mode=2 时选择式（3.2）。

如图 3–2 所示，（a）是原始图像，（b）是阈值 100 以下，（c）是阈值 128 以上，（d）是阈值 100～128 提取后反转。由于图 3–2 的图像中钳子本身有比背景亮的部分，还有比背景暗的部分，所以无论是使用式（3.1）还是使用式（3.2）都会丢失钳子的一部分，如图 3–2(b) 和图 3.2(c) 所示。在这种情况下，阈值处理可以通过提取两个阈值之间的部分来实现，如式（3.3）所示。

$$g(x,y)=\begin{cases}HIGH & t_1\leq f(x,y)\leq t_2\\ LOW & \text{其他情况}\end{cases} \tag{3.3}$$

式（3.3）的处理程序表示在 List 3.2 中。用 List 3.2 程序对原始图像处理以后，对处理结果再进行反转，可获得如图 3–2（d）的结果。反转程序显示在 List 3.3 中。这种方法被称为双阈值二值化处理。

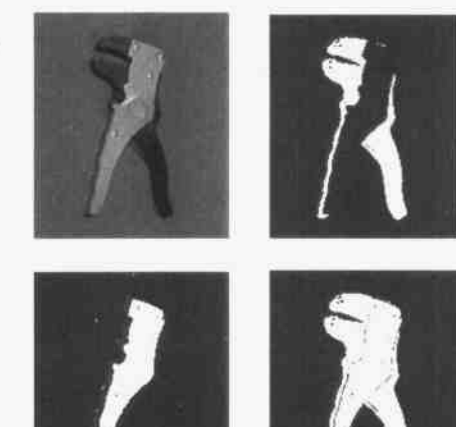


图 3-2



阈值的设定对二值化处理后产生的结果影响非常大。从图 3–3 中的二值图像可以发现，过小的阈值会把不需要的内容（例如背景）也一起提取出来了，相反，过大的阈值会去掉部分目标物体（米粒），其中（a）是原始图像，（b）是阈值=30，（c）是阈值=70，（d）是阈值=110。所以确定最佳阈值十分重要。

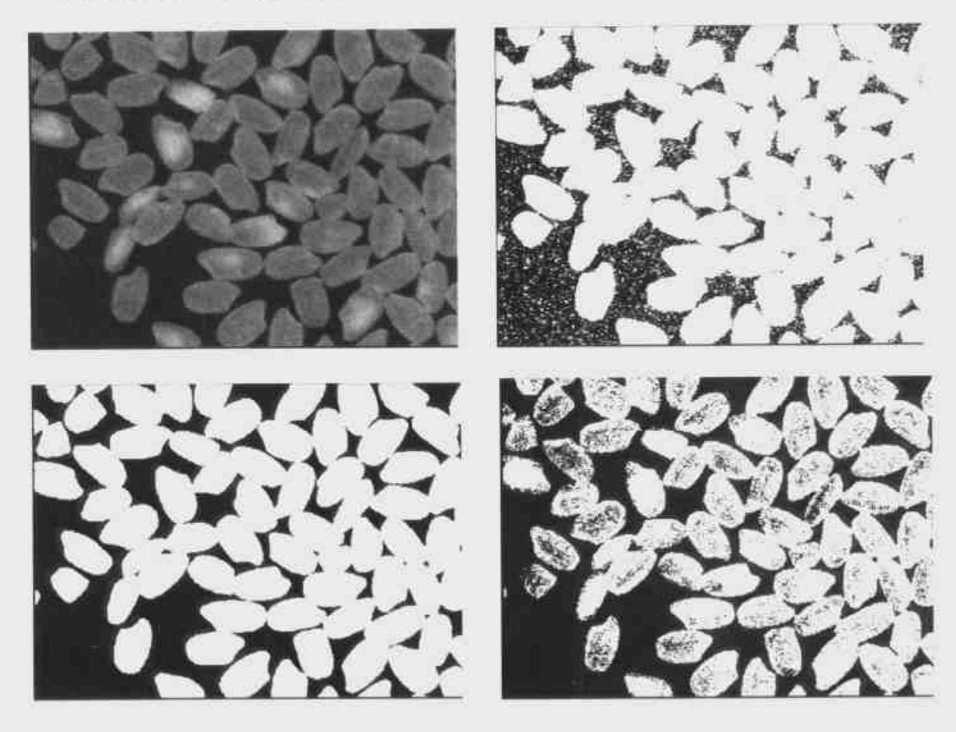


图 3-3

3.3 阈值的确定

要提取的物体的灰度与背景的灰度之间应该存在一定的区别，否则，人的眼睛也将无法识别。为此，在一些点或位置上查看需要提取的部分与想要去除的部分的灰度。在图 3–3 的例子中，米粒的灰度值约在 70～180 之间，背景部分约在 0～70 之间，从而选阈值 70 就能将米粒与背景分开。

然而，有没有一个快捷获取各个像素灰度值的方法呢？答案是肯定的，那就是使用直方图（频度分布）的方法。直方图（histogram）如图 3–4 所示，表示灰度值 i 的像素在画面中有多少个，可以由 List 3.4 所示的程序来求得。



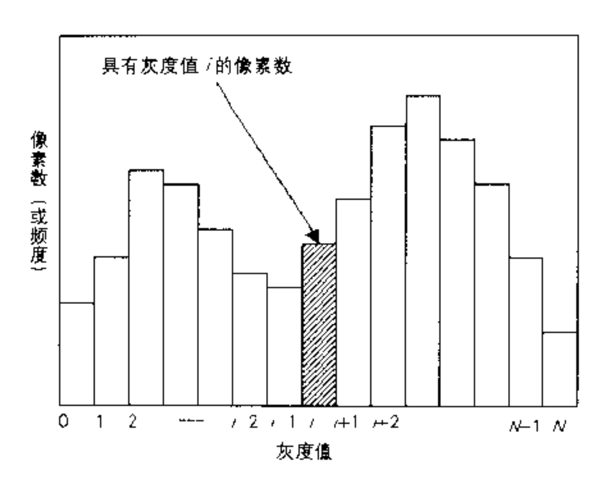


图 3-4

用 List 3.4 把图 3-3 所示图像的直方图显示在图 3–5 中。灰度值 120 左右的波峰相当于米粒部分，灰度值 20 左右的波峰相当于背景的像素。这两个波峰的交接处，即把波谷之处的值 70 取作阈值的话，可以准确地分离米粒和背景。

图 3–5 表示的是 Visual C++界面的直方图，对 Visual C++不太熟练的初学者来说，实现 Visual C++界面的直方图不是一件容易的事。为此，添加了一个直方图图像表示的程序 List 3.5，初学者可以输入该程序，简单地实现直方图的图像表示。在该程序中，设定图像的宽度在 64 像素以下时不进行处理，因为如果图像太小，就不能利用它的直方图进行物体判别。

然而，如图 3–6(a)所示的原始图像的直方图（图 3–6(b)所示），其凹凸激烈，难于确定波谷的位置。为了便于发现波谷，经常采取在直方图上对邻域点平均化的方法，以减少直方图的凹凸不平。List 3.6 列出了采用邻域 5 点对直方图进行平滑处理的程序，该程序把灰度值 $i-2$、$i-1$、i、$i+1$、$i+2$ 的频度的平均值作为平滑后灰度值 i 的频度。用该平滑化程序，对图 3–6(b)的直方图进行处理，得到图 3–6(c)所示的直方图。在平滑后的直方图上，可以很容易看出波谷处于灰度值 50 附近。选用 50 作为阈值，进行阈值处理的结果如图 3–6(d)所示，其中，(a)是原始图像，(b) 是直方图，(c) 是平滑化直方图，(d) 是阈值处理结果。虽然原始图像的照明偏暗，但是结果比较好，枝条几乎全部从背景中提取了出来。像这样取直方图的波谷作为阈值的方法称为模态法（mode method）。

在阈值确定方法中除了模态法以外，还有 p 参数法（p–tile method）、判别分析法（discriminant analysis method）、可变阈值法（variable thresholding）等。p 参数法是指当物体占整个图像的比例已知时（如 p%），在直方图上，从暗灰度一侧起（或者亮灰度一侧起）的累计像素数占总像素数 p%的地方的灰度值作为阈值。判别分析法是当在直方图分成物体和背景两部分时，



利用两部分统计量的不同来确定阈值的方法。可变阈值法是在背景灰度多变的情况下使用的，对图像的不同部位设置不同的阈值。

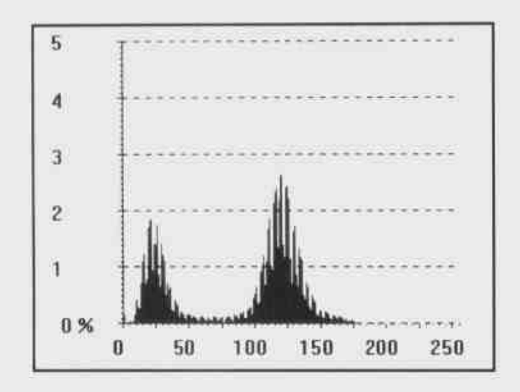
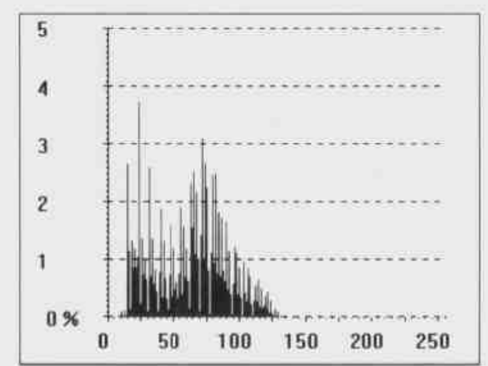


图 3-5

(a) | (b)
(c) | (d)





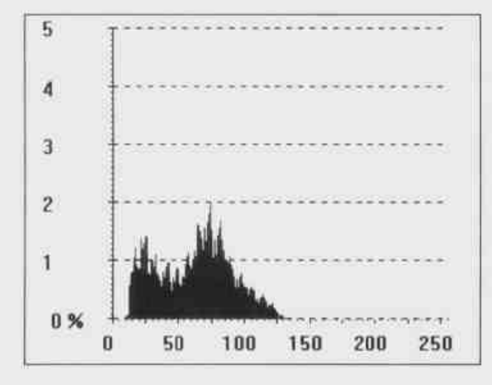




图 3-6



List 3.1　二值化处理

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Threshold --- 二值化处理 -----------------------------
   image_in:   输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   thresh:     阈值(0 - 255)
   mode:       处理方法(1,2)
-----------------------------------------------------------*/
void Threshold(BYTE *image_in, BYTE *image_out, int xsize, int ysize, int thresh,
int mode)
{
   int i,j;

   for (j = 0; j < ysize; j++)
   {
     for ( i = 0; i < xsize; i++)
     {
         switch (mode)
         {
           case 2:
               if (*(image_in +j*xsize + i) <= thresh)
                       *(image_out +j*xsize + i) = HIGH;
               else    *(image_out +j*xsize + i) =  LOW;
               break;
           default:
               if (*(image_in +j*xsize + i) >= thresh)
               *(image_out +j*xsize + i) = HIGH;
           else    *(image_out +j*xsize + i) =  LOW;
         break;
       }
     }
   }
}
```



List 3.2 双阈值二值化处理

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Threshold_mid --- 双阈值二值化处理 ----------------------------
   image_in:      输入图像数据指针
   image_out:     输出图像数据指针
   xsize:         图像宽度
   ysize:         图像高度
   thresh_low:    低阈值(0 - 255)
   thresh_high:   高阈值(0 - 255)
------------------------------------*/
void Threshold_mid(BYTE *image_in, BYTE *image_out, int xsize,
        int ysize, int thresh_low, int thresh_high)
{
   int i,j;

   for (j = 0; j < ysize; j++)
   {
     for ( i = 0; i < xsize; i++)
     {
       if (*(image_in +j*xsize + i) >= thresh_low &&
         *(image_in +j*xsize + i) <= thresh_high)
         *(image_out +j*xsize + i) = HIGH;
       else    *(image_out +j*xsize + i) =  LOW;

     }
   }
}
```

List 3.3 图像反转

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Reverse_image --- 反转图像数据 ----------------------------------
   image_in:    输入图像数据指针
   image_out:   输出图像数据指针
   xsize:       图像宽度
```



```
   ysize:      图像高度
- ------------------------------------*/
void Reverse_image(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
   int i,j;

   for (j = 0; j < ysize; j++)
   {
     for ( i = 0; i < xsize; i++)
     {
       *(image_out +j*xsize + i) = HIGH - *(image_in +j*xsize + i);
     }
   }
}
```

List 3.4 直方图

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Histgram --- 灰度分布直方图 ----------------------------------   ---
     image:  图像数据指针
     xsize:  图像宽度
     ysize:  图像高度
     hist:   直方图配列
-----------------------------------------------------------   --------*/
void Histgram(BYTE *image, int xsize, int ysize, long hist[256])
{
int i,j,n;

  for (n = 0; n < 256; n++) hist[n] = 0;
  for (j = 0; j < ysize; j++) {
       for ( i = 0; i < xsize; i++) {
           n = *(image +j*xsize + i);
           hist[n]++;
       }
   }
}
```



List 3.5　直方图图像化

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Hist_to_image --- 直方图图像化 (xsize 大于等于 64) ----
   hist:        直方图数据
   image_hist:  直方图图像
   xsize:       图像宽度
   ysize:       图像高度
---------------------------------------------------------------*/
void Hist_to_image(long hist[256], BYTE* image_hist, int xsize, int ysize)
{
   int   i, j, k, max, range;
   long  n;
   float d;

   if(xsize < 64) return;

   range = ysize - 5;
   for (j = 0; j < ysize; j++) {
     for (i = 0; i < xsize; i++) {
       *(image_hist + j*xsize +i) = LOW;
     }
   }
   if (xsize >= 256) {
     max = 0;
     for (i = 0; i < 256; i++) {
       n = hist[i];
       if (n > max) max = n;
     }
     for (i = 0; i < 256; i++) {
       d = (float)hist[i];
       n = (long)(d / (float)max * (float)range);
       for(j=0; j <= n; j++) *(image_hist + (range -j)*xsize + i) = HIGH;
     }
     for (i= 0; i<= 4; i++) {
       k = 64 * i;
       if (k >= xsize) k = xsize - 1;
```



```
      for (j= range; j< ysize; j++) *(image_hist + j*xsize + k) = HIGH;
    }
  }
  else if (xsize >= 128) {
    max = 0;
    for (i = 0; i < 128; i++) {
      n = hist[2*i] + hist[2*i+1];
      if (n > max) max = n;
    }
    for (i = 0; i < 128; i++) {
      d = (float)(hist[2*i]+hist[2*i+1]);
      n = (long)(d / (float)max * (float)range);
      for(j = 0; j <= n; j++) *(image_hist + (range-j)*xsize + i) = HIGH;
    }
    for (i= 0; i<= 4; i++) {
      k = 32 * i;
      if (k >= xsize) k = xsize - 1;
      for (j= range; j< ysize; j++) *(image_hist + j*xsize +k) = HIGH;
    }
  }
  else if (xsize >= 64) {
    max = 0;
    for (i = 0; i < 64; i++) {
      n = hist[4*i] + hist[4*i+1] + hist[4*i+2] + hist[4*i+3];
      if (n > max) max = n;
    }
    for (i = 0; i < 64; i++) {
      d = (float)(hist[4*i] + hist[4*i+1] + hist[4*i+2] + hist[4*i+3]);
      n = (long)(d / (float)max * (float)range);
      for(j = 0; j <= n; j++) *(image_hist + (range-j)*xsize + i) = HIGH;
    }
    for (i= 0; i<= 4; i++) {
      k = 16 * i;
      if (k >= xsize) k = xsize - 1;
      for (j= range; j< ysize; j++) *(image_hist + j*xsize + k) = HIGH;
    }
  }
```



```
}
```

List 3.6 直方图平滑化

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Histsmooth --- 直方图平滑化 --------------------------------
  hist_in:  输入直方图配列
  hist_out: 输出直方图配列
-----------------------------------------------------------------*/
void Hist_smooth(long hist_in[256],long hist_out[256])
{
  int  m, n, i;
  long sum;

  for (n = 0; n < 256; n++) {
    sum = 0;
    for (m = -2; m <= 2; m++) {
      i = n + m;
      if (i <   0) i =   0;
      if (i > 255) i = 255;
      sum = sum + hist_in[i];
    }
    hist_out[n] = (long)((float)sum / 5.0 + 0.5);
  }
}
```



第4章

边缘检测与提取

4.1 边缘与图像处理

在图像处理中，边缘（edge）（或称轮廓，contour）不仅仅是指物体边界的线，还应该包括能够描绘图像特征的线要素，这些线要素就相当于素描画中的线条。当然，除了线条以外，颜色以及亮度也是图像的重要因素，但是日常所见到的说明图、图表、插图、肖像画、连环画等，很多是通过描绘物体的边缘线来表现的，尽管有些单调，我们还是能够非常清楚地明白图像的内容。这似乎有点不可思义，简单的边缘线就能让人们理解所要表述的物体。对于图像处理来说，边缘检测（edge detection）也是重要的基本操作之一。利用所提取的边缘可以识别出特定的物体、测量物体的面积及周长、求两幅图像的对应点等，所以边缘检测与提取处理是复杂的图像识别、图像理解的关键。

4.2 边缘性质的描述

边缘与图像的性质之所以能联系在一起，是由于图像中的物体与物体，或者物体与背景之间的交界是边缘，图像的灰度及颜色急剧变化的地方可以看作是边缘。由于自然图像中颜色的变化必定会有灰度的变化，因此对于边缘的检测与提取，只要把焦点集中在灰度上就可以了。

图像边缘的灰度变化模型如图 4—1 所示。图 4—1（a）表示的是阶梯型边缘的灰度变化，这是一个典型的模式，可以很明显地看出边缘，也称之为轮廓。物体与背景的交界处会产生这种阶梯状的灰度变化。图 4—1（b）是线条本身的灰度变化，当然这个也是很明显的边缘。线条状的物体，在照明程度不同的情况下，使物体上带有阴影等情况，都能产生线条型边缘。在图 4—1（c）中有灰度变化，但变化平缓，边缘不明显。如图 4—1（d）所示是灰度以折线状变化的，这种情况不如 4—1（b）明显，但折线的角度变化急剧，还是能看出边缘的。

在图 4—2 中，显示的是人物照片边缘部分的灰度分布，十分清楚的边缘不是阶梯状，有些变钝了，呈现出斜坡状，即使是同一物体的边缘，因为地点的不同，灰度变化也不同，可以观察到边缘存在着模糊部分。由于大多数传感元件具有低频特性，从而使阶梯型边缘变成斜坡型边缘，线条型边缘变成折线型边缘，这是不可避免的。

因此，在实际图像中（不包括由计算机图形学制作出的图像），即使用眼睛可以清楚地确定边缘，在灰度变化模型中也有一些会变钝、灰度变化量会变小，从而使得提取清晰的边缘变得十分困难，因此人们提出了各种各样的算法。



(a) (b)
(c) (d)

图 4-1

图 4-2

4.3 基于微分的边缘检测与提取

由于边缘为灰度值急剧变化的部分，所以微分作为提取函数变化部分的运算能够在边缘检测与提取中利用。微分运算中有一阶微分（first differential calculus）（也称 gradient，梯度运算）与二阶微分（second differential calculus）（也称 Laplacian，拉普拉斯运算），这两种微分都可以用在边缘检测与提取中。





(1) 在窗体 Form1 上放一个名为 Combobox1 组合框和三个命令按钮，将三个命令按钮的 Text 属性分别设置为“删除订单”、“将订单号加入到 Combobox1”和“字段名加入到 Combobox1”。

(2) 在窗体的代码窗口定义连接变量 con 和记录集变量 rs。

```
Dim con As New ADODB.Connection()
Dim rs As New ADODB.Recordset()
```

(3) 定义连接数据库的过程 opendb()。

```
Sub opendb()
  '使用 OLE DB 连接到 SQL
 con.Open("Provider=SQLOLEDB.1;Integrated Security=SSPI; " _
& "Persist Security Info=False;" Database=Northwind")
End Sub
```

(4) 在完成“删除订单”的命令按钮的 Click 事件过程中加入代码。

```
If IsNumeric(Combobox1.Text) Then     '判断 Combobox1 中是否是数字
opendb()
con.Execute("delete from [order details] where orderid=" & Combobox1_
.Text)
 MsgBox("删除完毕！")
con.Close()
End If
```

(5) 在完成“将订单号加入到 Combobox1”的命令按钮的 Click 事件过程中加入代码。

```
opendb()
rs.Open("orders", con, ADODB.CursorTypeEnum.adOpenKeyset)
Combobox1.Items.Clear()
Do While Not rs.EOF
Combobox1.Items.Add(rs(0).Value)
rs.MoveNext()
Loop
con.Close()
```

(6) 在完成“将字段名加入到 Combobox1”的命令按钮的 Click 事件过程中加入代码。

```
opendb()
Dim i As Integer
Combobox1.Items.Clear()
rs.Open("select * from orders", con, ADODB.CursorTypeEnum.adOpenKeyset)
```



f_y的计算式（4.1），以及拉普拉斯运算的式（4.4），都是基于这些微分算子而进行微分运算的。这些微分算子如表4−1、表4−2所示，其中有多个种类。实际的微分运算，就是计算目标像素及其周围像素，分别乘上微分算子对应数值矩阵系数的和，其计算结果被用作微分运算后目标像素的灰度值

表4-1 采用梯度计算的微分算子

算子名称	一般差分	Roberts 算子	Sobel 算子
	0 0 0	0 0 0	−1 0 1
求 f_x 的模板	0 1 1	0 1 0	−2 0 2
	0 0 0	0 0 −1	−1 0 1
	0 0 0	0 0 0	−1 −2 −1
求 f_y 的模板	0 1 0	0 0 1	0 0 0
	0 −1 0	0 −1 0	1 2 1

表4-2 采用拉普拉斯计算的微分算子

算子名称	拉普拉斯算子1	拉普拉斯算子2	拉普拉斯算子3
	0 −1 0	−1 −1 −1	1 −2 1
模　板	−1 4 −1	−1 8 −1	−2 4 −2
	0 −1 0	−1 −1 −1	1 −2 1

使用微分算子进行边缘检测与提取的程序表示在 List 4.1（gradient operator，梯度算子）和 List 4.2（Laplacian operator，拉普拉斯算子），在练习时可以将上述算子分别代入程序，观察处理结果。另外，对于拉普拉斯算子，由于计算值以0为中心正负波动，所以取其绝对值作为输出像素值，输出图像是强调了边缘的灰度图像。程序中的变量 amp 是用于调整输出的图像强度，适当增加 amp 值可以使输出的图像看起来比较清晰，但是由于使用的微分算子的不同，提取的边缘强度也不同，因此 amp 值需要根据不同算子进行调整。

4.4 基于模板匹配的边缘检测与提取

模板匹配（template matching）就是研究图像与模板（template）的一致性（匹配程度）。为此，准备了几个表示边缘的标准模式，与图像的一部分进行比较，选取最相似的部分作为结果图像。

首先要考查一下其中的 Prewitt 算子。

如图4−4所示，准备了对应于8个边缘方向的8种掩模（mask）。图4−5说明了这些掩模与实际图像如何进行比较。与微分运算相同，目标像素及其周围像素分别乘以对应掩模的系



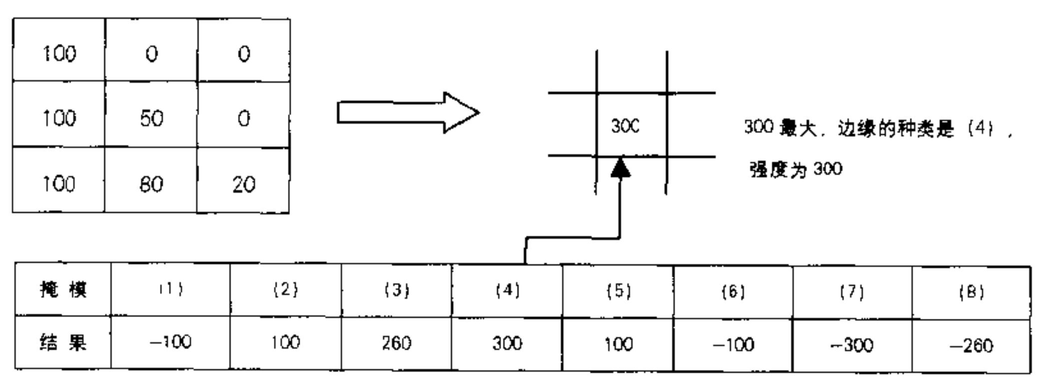
数值，然后将所有积求和。对 8 个掩模分别进行计算，其中计算结果值最大的掩模方向即为边缘的方向，其计算结果即为边缘的强度。

基于 Prewitt 算子的边缘检测与提取的程序显示在 List 4.3 中。输出结果是对应边缘强度的灰度图像。

此外，在同样的模板匹配中，有著名的 Hueckel 算子。它是把边缘和线等都模型化，与实际图像相比，是计算出一致模型参数的方法，但是稍微要复杂一些，在此不对其作详细说明。

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
掩模模式	1 1 1 1 −2 1 -1 −1 −1	1 1 1 1 2 −1 1 −1 −1	1 1 1 1 -2 −1 1 1 −1	1 −1 −1 1 −2 −1 1 1 1	−1 −1 -1 1 2 1 1 1 1	−1 -1 1 −1 −2 1 1 1 1	−1 1 1 −1 −2 1 −1 1 1	1 1 1 −1 -2 1 -1 −1 1
所对应的边缘	明 暗	明 暗	明 暗	暗 明	暗 明	暗 明	暗 明	明 暗

图 4-4



掩 模	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
结 果	−100	100	260	300	100	−100	−300	−260

（对于当前像素的 8 邻域，计算各掩模的一致程度）

例如，掩模(1) $1\times100+1\times0+1\times0+1\times100+(-2)\times50+1\times0+(-1)\times100+(-1)\times80+(-1)\times20=-100$

图 4-5

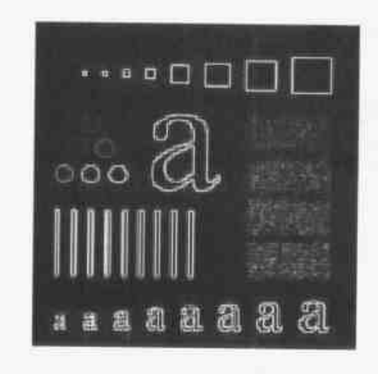
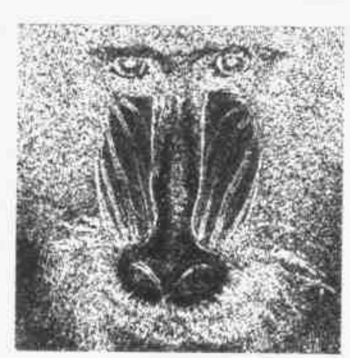
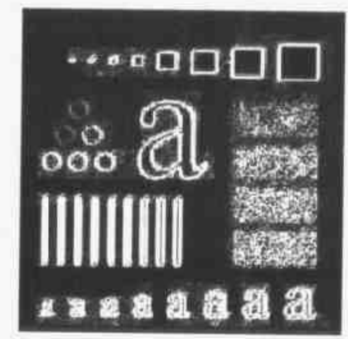


4.5 边缘检测与提取的实例

图 4-6 是采用上述方法的处理结果，其中，(a)是原始图像，(b)是 1 阶微分（Roberts 算子），(c)是 2 阶微分（拉普拉斯算子 2），(d)是模板匹配（Prewitt 算子）。由于参数 *amp* 具有同一数值，根据算子不同，有明暗方面的差异，但是在原始图像上边缘明显的部分都被清晰地提取出来了。

(a)
(b)
(c)
(d)







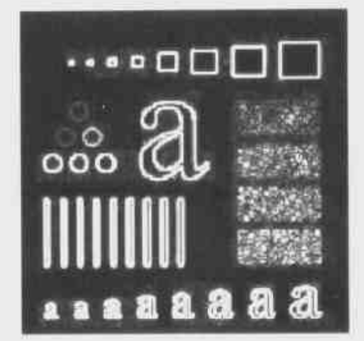
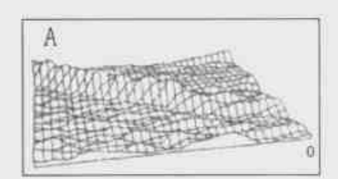


图 4-6

拉普拉斯算子易于强化噪声，可以说比起边缘检测与提取（特别对平缓的边缘）更适于点状物的检测与提取，以及图像的增强（使边缘清晰的图像处理——锐化）等。

在图 4–7 中表示了所提取的边缘的灰度变化，可以与图 4–2 比较一下。





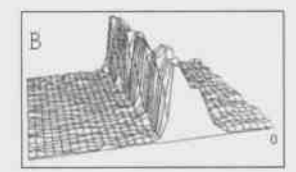


图 4-7

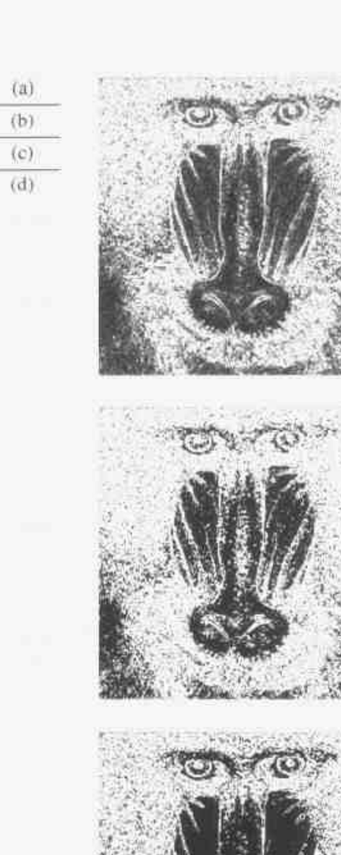
4.6 二值边缘图像的制作

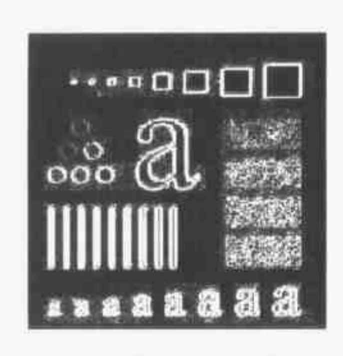
到目前为止叙述的所有方法，由于其输出图像是对应于边缘强度的灰度图像，如果要表示边缘线或者在打印机上打印，有必要进行二值化处理。所以，在边缘检测与提取处理时，在处理后的图像上进行阈值处理即可。

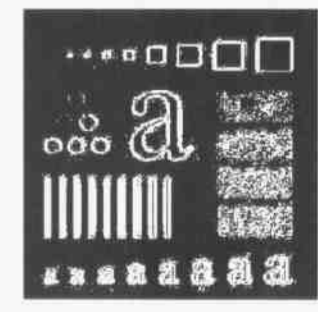
图 4–8 表示了二值边缘图像的例子，其中，(a)是输入的边缘图像（拉普拉斯算子 2），(b)是二值化图像(阈值 100 以上)，(c)是二值化图像(阈值 150 以上)，(d)是二值化图像(阈值 200 以上)。增大阈值，边缘线变得模糊不清或消失，相反地，如果减小阈值，不需要的噪声就会变多。除了经过反复设定来确定合适阈值之外，还可使用直方图的方法。另外，可以把第 5 章中将要介绍的膨胀腐蚀处理用于二值边缘图像，去除不需要的点状噪声。

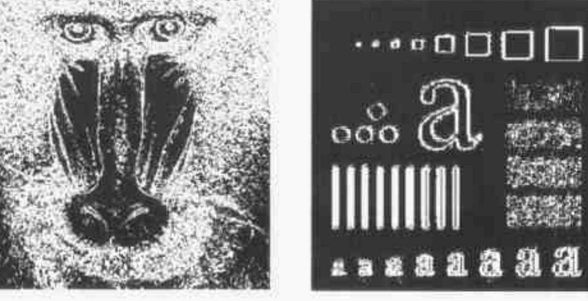


(a)
(b)
(c)
(d)









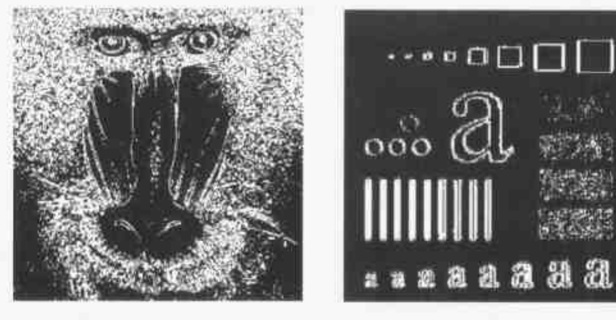


图 4-8



4.7 细线化处理

对使用上述方法所得到的边缘，再加以处理就可以获得清晰的边缘图像。在此，从边缘的精细化方法中，选取细线化（thinning）作简单介绍。

细线化是把线宽不均匀的边缘线整理成同一线宽（一般为 1 像素宽）的方法，在阈值处理后的二值图像上进行。

细线化处理，如图 4-9 那样，将粗边缘线从外侧开始逐层地削去各个像素，直到成为 1 像素的宽度为止。其中，（a）是原始图像，（b）是 1 次处理，（c）是 2 次处理，（d）是最终结果。根据削去像素的规则不同而有不同的方法，其中 Hilditch 算法表示在 List 4.4 中。图 4-10（b）是对输入的二值边缘图像（图 4-10（a））的处理结果，得到了线宽为 1 个像素的边缘图像。

(a) | (b)
(c) | (d)

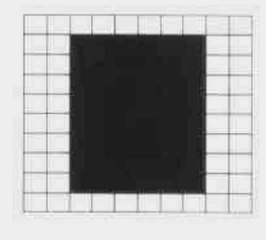
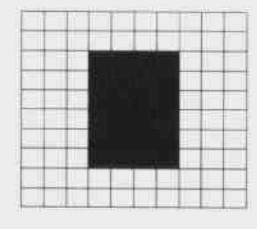
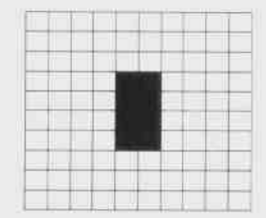
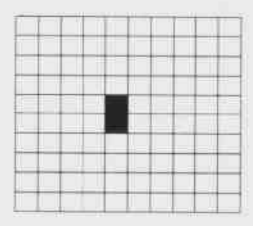





图 4-9

(a)
(b)

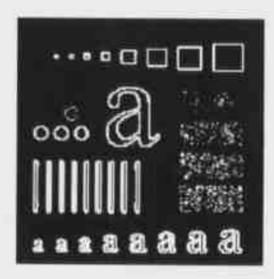





图 4-10

List 4.1　一阶微分边缘检测（梯度算子）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>

/*--- Differential ---1 阶微分边缘检测 ---------------------------------
   image_in: 输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   amp:      输出像素值倍数
-----------------------------------------------------------------*/
void Differential(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float amp)
{
   //以下算子可以自由设定
   static int cx[9] = { 0, 0, 0, //算子 x(Roberts)
            0, 1, 0,
            0, 0,-1};
   static int cy[9] = { 0, 0, 0, //算子 y(Roberts)
            0, 0, 1,
            0,-1, 0};
   int   d[9];
   int   i, j, dat;
   float xx, yy, zz;

   for (j = 1; j < ysize-1; j++) {
      for (i = 1; i < xsize-1; i++) {
       d[0] = *(image_in + (j-1)*xsize + i-1);
```

```
        d[1] = *(image_in + (j-1)*xsize + i);
        d[2] = *(image_in + (j-1)*xsize + i+1);
        d[3] = *(image_in + j*xsize + i-1);
        d[4] = *(image_in + j*xsize + i);
        d[5] = *(image_in + j*xsize + i+1);
        d[6] = *(image_in + (j+1)*xsize + i-1);
        d[7] = *(image_in + (j+1)*xsize + i);
        d[8] = *(image_in + (j+1)*xsize + i+1);
        xx = (float)(cx[0]*d[0] + cx[1]*d[1] + cx[2]*d[2]
             + cx[3]*d[3] + cx[4]*d[4] + cx[5]*d[5]
             + cx[6]*d[6] + cx[7]*d[7] + cx[8]*d[8]);
        yy = (float)(cy[0]*d[0] + cy[1]*d[1] + cy[2]*d[2]
              + cy[3]*d[3] + cy[4]*d[4] + cy[5]*d[5]
              + cy[6]*d[6] + cy[7]*d[7] + cy[8]*d[8]);
        zz = (float)(amp*sqrt(xx*xx+yy*yy));
        dat = (int)zz;
        if(dat > 255) dat = 255;
        *(image_out + j*xsize + i) = dat;
      }
    }
}
```

List 4.2　二阶微分边缘检测（拉普拉斯算子）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>

/*--- Differential --- 2阶微分边缘检测 ---------------------------------
   image_in:    输入图像数据指针
   image_out:   输出图像数据指针
   xsize:       图像宽度
   ysize:       图像高度
   amp:         输出像素值倍数
----------------------------------------------------------------------*/
void Differential2(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float amp)
{
   //以下算子可以自由设定
```



```
    static int c[9] = {-1, -1, -1 // 算子(laplacian)
            -1, 8, -1
            -1, -1, -1};
    int d[9];
    int i, j, dat;
    float z, zz;

    for (j = 1; j < ysize-1; j++) {
       for (i = 1; i < xsize-1; i++) {
        d[0] = *(image_in + (j-1)*xsize + i-1);
        d[1] = *(image_in + (j-1)*xsize + i);
        d[2] = *(image_in + (j-1)*xsize + i+1);
        d[3] = *(image_in + j*xsize + i-1);
        d[4] = *(image_in + j*xsize + i);
        d[5] = *(image_in + j*xsize + i+1);
        d[6] = *(image_in + (j+1)*xsize + i-1);
        d[7] = *(image_in + (j+1)*xsize + i);
        d[8] = *(image_in + (j+1)*xsize + i+1);
        z =  (float)(c[0]*d[0] + c[1]*d[1] + c[2]*d[2]
              + c[3]*d[3] + c[4]*d[4] + c[5]*d[5]
              + c[6]*d[6] + c[7]*d[7] + c[8]*d[8]);
        zz = amp*z;
        dat = (int)(zz);
        if (dat <  0) dat = -dat;
        if (dat > 255) dat =  255;
        *(image_out + j*xsize + i) = dat;
      }
    }
}
```

List 4.3　Prewitt算子边缘检测

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
/*---Prewitt--- Prewitt 算子边缘检测 ------------------------------------
    image_in:    输入图像数据指针
    image_out:   输出图像数据指针
```



```
   xsize:       图像宽度
   ysize:       图像高度
   amp:         输出像素值倍数
-----------------------------------------------------------------------*/
void Prewitt(BYTE*image_in,BYTE *image_out,int xsize,int ysize,float amp)
{
   int   d[9];
   int   i,j,k,max,dat;
   int   m[8];
   float zz;

   for (j = 1; j < ysize-1; j++) {
      for (i = 1; i < xsize-1; i++) {
         d[0] = *(image_in + (j-1)*xsize - i-1);
         d[1] = *(image_in + (j-1)*xsize - i);
         d[2] = *(image_in + (j-1)*xsize + i+1);
         d[3] = *(image_in + j*xsize + i-1);
         d[4] = *(image_in + j*xsize + i);
         d[5] = *(image_in + j*xsize + i+1);
         d[6] = *(image_in + (j+1)*xsize + i-1);
         d[7] = *(image_in + (j+1)*xsize + i);
         d[8] = *(image_in + (j+1)*xsize + i+1);
      m[0] =  d[0] + d[1] + d[2] + d[3] -2*d[4] + d[5] - d[6] - d[7] - d[8];
      m[1] =  d[0] + d[1] + d[2] + d[3] -2*d[4] - d[5] + d[6] - d[7] - d[8];
      m[2] =  d[0] + d[1] - d[2] + d[3] -2*d[4] - d[5] + d[6] + d[7] - d[8];
      m[3] =  d[0] - d[1] - d[2] + d[3] -2*d[4] - d[5] + d[6] + d[7] + d[8];
      m[4] = -d[0] - d[1] - d[2] + d[3] -2*d[4] + d[5] + d[6] + d[7] + d[8];
      m[5] = -d[0] - d[1] + d[2] - d[3] -2*d[4] + d[5] + d[6] + d[7] + d[8];
      m[6] = -d[0] + d[1] + d[2] - d[3] -2*d[4] + d[5] - d[6] + d[7] + d[8];
      m[7] =  d[0] + d[1] + d[2] - d[3] -2*d[4] + d[5] - d[6] - d[7] + d[8];
         max = 0;
         for (k = 0; k < 8; k++) if (max < m[k]) max = m[k];
      zz = amp*(float)(max);
         dat = (int)(zz);
         if (dat > 255) dat = 255;
         *(image_out + j*xsize + i) = dat;
     }
```



```
    }
}
```

List 4.4 二值图像的细线化处理

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
int cconc(int inb[9] );
/*--- Thinning --- 2 值图像的细线化处理 ----------------------------------
    image_in:   输入图像数据指针
    image_out:  输出图像数据指针
    xsize:      图像宽度
    ysize:      图像高度
------------------------------------------------------------------------*/

void Thinning(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
    int ia[9], ic[9], i, ix, iy, m, ir, iv, iw;

    for (iy = 0; iy < ysize; iy++)
      for (ix = 0; ix < xsize; ix++)
        *(image_out + iy*xsize +ix) = *(image_in + iy*xsize +ix);
    m = 100;
    ir = 1 ;
    while (ir != 0) {
      ir = 0;
      for (iy = 1; iy < ysize-1; iy++)
        for (ix = 1; ix < xsize-1; ix++) {
          if ( *(image_out + iy*xsize + ix) != HIGH) continue;
          ia[0] = *(image_out +iy*xsize + ix+1);
          ia[1] = *(image_out +(iy-1)*xsize + ix+1);
          ia[2] = *(image_out +(iy-1)*xsize + ix);
          ia[3] = *(image_out +(iy-1)*xsize + ix-1);
          ia[4] = *(image_out + iy*xsize + ix-1);
          ia[5] = *(image_out + (iy+1)*xsize + ix-1);
          ia[6] = *(image_out + (iy+1)*xsize + ix);
          ia[7] = *(image_out + (iy+1)*xsize + ix+1);
```



```
        for (i = 0; i < 8; i++) {
          if (ia[i] == m) {
            ia[i] = HIGH; `
            ic[i] = 0;
          }
          else {
            if (ia[i] < HIGH) ia[i] = 0;
            ic[i] = ia[i];
          }
        }
        ia[8] = ia[0];
        ic[8] = ic[0];
        if (ia[0]+ia[2]+ia[4]+ia[6] == HIGH*4) continue;
        for (i = 0, iv = 0, iw = 0; i < 8; i++) {
          if (ia[i] == HIGH) iv++;
          if (ic[i] == HIGH) iw++;
        }
        if (iv <= 1) continue;
        if (iw == 0) continue;
        if (cconc(ia) != 1) continue;
              if ( *(image_out + (iy-1)*xsize + ix) == m) {
          ia[2] = 0;
          if (cconc(ia) != 1) continue;
          ia[2] = HIGH;
        }
        if ( *(image_out + iy*xsize + ix-1) == m) {
          ia[4] = 0;
          if (cconc(ia) != 1) continue;
          ia[4] = HIGH;
        }
        *(image_out + iy*xsize + ix) = m;
        ir++;
      }

    m++;
  }
  for (iy = 0; iy < ysize; iy++)
```



```
  for (ix = 0; ix < xsize; ix++)
    if(*(image_out+iy*xsize+ix)<HIGH)*(image_out+iy*xsize+ix)=0;
}

/*--- cconc --- 计算连接数 ---------------------------------------
(In) inb:  连接数
---------------------------------------------------------------*/
int cconc(int inb[9])
{
  int i, icn;
  icn = 0;

  for (i = 0; i < 8; i += 2)
    if (inb[i] == 0)
    if (inb[i+1] == HIGH || inb[i+2] == HIGH)
    icn++;
    return icn;
}
```



```
TextBox1.Text = rs1("customerid").Value
TextBox2.Text = rs1("employeeid").Value
TextBox3.Text = rs1("orderdate").Value
```

程序运行后，从“组合框”控件中选择订单号后，单击“查询”按钮，显示的界面如图 4-14 所示。

图 4-14

4.3 ADO 在数据库中的应用

在学习了 VB.NET 中 ADO 内容后，对 ADO 已经有了相当的了解，下面综合前边介绍的 ADO 各对象，介绍其在数据库中的应用。

本示例以 Student.MDB 中的 Student 表为例（表结构见第 12 章），说明如何应用 ADO 增加、删除、查找记录，示例中主要应用了 Connection、Command、Recordset 及 Parameter 对象。

程序设计界面如图 4-15 所示。

图 4-15



5.1 关于图像噪声

噪声（noise）在这里指障碍物。那么，图像的噪声是什么呢？例如电视机天线的状况不佳，图像变得混乱，这样的状态被称为图像的劣化。这种图像劣化可以分成两类，一类是目标图像本身变形或模糊而劣化，另一类是粗糙的障碍物附在目标图像上而劣化。像后者那样的障碍物就是图像的噪声（杂质）。根据噪声的性质不同，消除噪声的方法也有所不同。图 5-1 是一个带有噪声的图像的例子，这种噪声具有以下性质：

- 在画面的何处附加噪声是随机的。
- 噪声的形状大小是不规则的。

这种噪声称为随机噪声（random noise），是一种线索最少却最常见的噪声。用摄像机在暗处拍摄的图像上会感觉像附有砂状物，这是具有随机噪声的图像的最好例子。



图 5-1

在此，考虑静止的相同物体被连续拍摄时，噪声会随时随机出现，每一个瞬间，噪声的位置和大小都有所不同。图 5-2 表示被输入多帧并具有随机噪声的图像。在这种情况下，可以利用噪声随时随机的性质，来有效地消除噪声。

现在来关注画面上的某像素，通过多帧图像研究该像素，噪声的灰度值应该如图 5-3 所示的那样以实际值（真值）为中心散落着。因此，取这些值的平均，研究的帧数越多应越接近实际值。它的原理是，如果使用无数帧的图像，噪声将基本被消除。



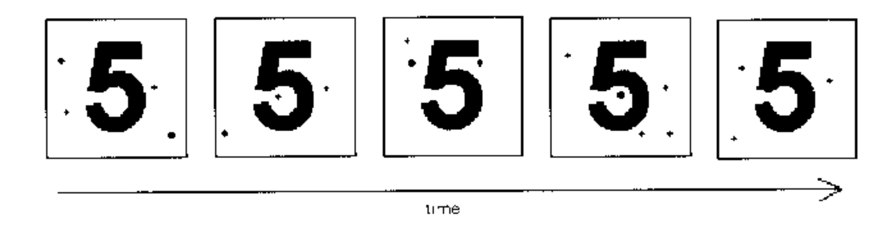


图 5-2

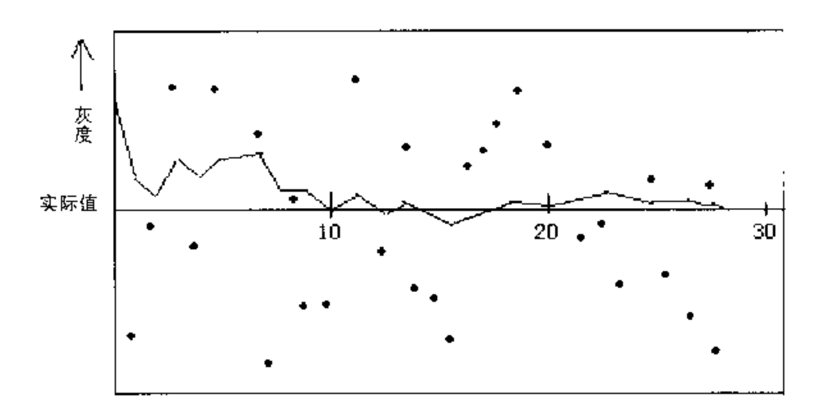


图 5-3

5.2 图像平滑

对只有一帧带有噪声的图像，要消除噪声的话，该如何处理呢？这就是本章的主题，这时由随机噪声隐藏的像素的实际灰度值是不可知的，此时的目的是使噪声不要成为障碍，在视觉上不明显就可以了。

下面来看一下带有噪声的放大图像，观察图 5–4 可以看出，噪声的灰度与其周围的灰度之间有着明显的灰度差，也正是这些明显的灰度差才造成了视觉障碍。一般情况下，把利用噪声的性质来消除图像中噪声的方法称为图像平滑（image smoothing）或简称为平滑（smoothing）。因为目标图像的边缘部分也具有明显的灰度差，所以如何把边缘部分与噪声部分区分开，是图像消除噪声的关键所在。

移动平均法（moving average model。或称 averaging filter，均值滤波器）是消除噪声最简单的方法。如图 5–5 所示，用某像素周围 3 × 3 像素范围的平均值置换该像素值。它的原理是，



通过使图像模糊，达到看不到细小噪声的目的。但是，使用这种方法时，不管噪声还是边缘都一起模糊化，结果是噪声被消除的同时，目标图像也变模糊了。



图 5-4

消除噪声最好的结果应该是，噪声被消除了，而边缘还完好地保留着。能达到这种处理效果的方法是中值滤波（median filter）。

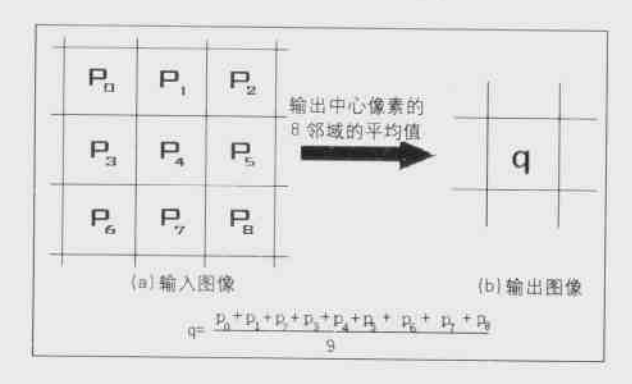


图 5-5

5.3 中值滤波

如图 5-6 所示的灰度图像的数据，为了求由○所围的像素值，查看 3×3 邻域内（黑框线所围的范围）的 9 个像素的灰度，按照从小到大的顺序排列，即 2、2、3、3、4、4、4、5、10。

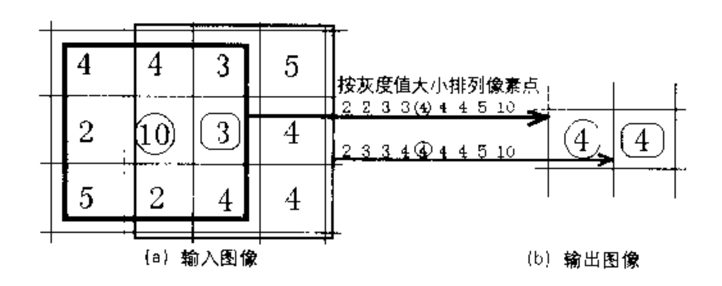
这时的中间值（也称 medium，中值）应该是排序后 9 个像素中第 5 个的灰度值 4。灰度值 10 的像素是作为噪声故意输入进去的，通过中值处理后被消除了。因为与周围像素相比噪声



的灰度值是相差很远的，按大小排序时它将集中在左端或右端。

那么，其右侧的像素（由▢所围的像素）又如何呢？查看一下细框线所围的邻域内的像素。同样按照从小到大的顺序排列，即2、3、3、4、4、4、4、5、10。

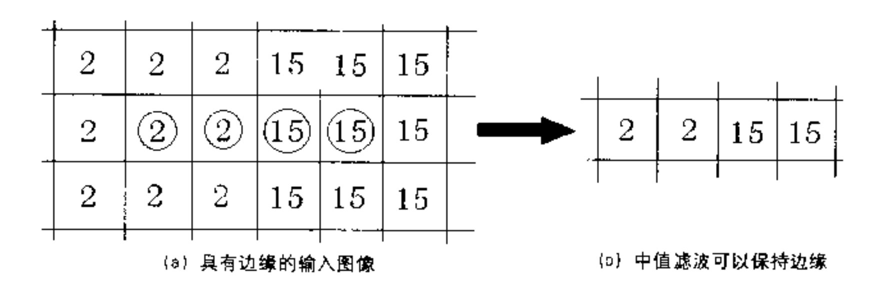
中间值是4，实际上是3却成了4。这是由于处理所造成的。但是，视觉上这是看不出来的。



(a) 输入图像　　(b) 输出图像

图5-6

图5-7（a）是具有边缘的图像，求由○所围的像素，得到图5-7（b）的结果，可见边缘被完全保存下来了。



(a) 具有边缘的输入图像　　(b) 中值滤波可以保持边缘

图5-7

在移动平均法中由于噪声成分被放入平均计算之中，所以输出受到了噪声的影响。但是在中值滤波中由于噪声成分很难被选上，所以几乎不会影响到输出。因此，用同样的3×3区域进行比较，中值滤波消除噪声的能力会更胜一筹。

举个例子，实际处理一幅图像看一下效果。图5-8表示用中值滤波和移动平均法除去噪声的结果，从中可以很清楚地看出中值滤波无论在消除噪声方面，还是在保存边缘方面都是一个不错的方法，其中，（a）是原始图像，（b）是中值滤波，（c）是移动平均法。但是，中值滤波花费的计算时间是移动平均法的5倍多。



(a)(b)(c)





图 5-8

移动平均法和中值滤波消除噪声的程序分别显示在 List 5.1 和 List 5.2 中。

5.4 二值图像的平滑

在第 3 章的区域生成和第 4 章的边缘检测中，输出图像都是二值图像。在进行这些处理之前，平滑处理消除噪声是很重要的。这被称为预处理（pre-processing），它使得后续处理更加容易。即使如此，预处理后输出的二值图像中还是会有一些噪声，这就需要在后续处理中消除。

二值图像的噪声如图 5-9 所示，被称为椒盐噪声，也就是英语 salt-and-pepper noise 的直译。当然这种噪声能够用中值滤波消除，但是由于它只有二值，也可以采用膨胀与腐蚀的处理方法来消除。



图 5-9

膨胀（dilation）是指某像素的邻域内，只要有一个像素是白像素则该像素就由黑变为白，其他保持不变；腐蚀（erosion）是指某像素的邻域内，只要有一个像素是黑像素则该像素就由白变为黑，其他保持不变。如图 5-10 所示，其中，(a)是膨胀（增粗了 1 像素，除去了黑色噪



声）、(b)是腐蚀（除去了黑色噪声，白色噪声还残留）、(c)是腐蚀（削减了 1 像素，除去了白色噪声）、(d)是膨胀(除去了白色噪声，黑色噪声还残留)。经过膨胀、腐蚀处理后，膨胀变粗，腐蚀变细，结果是图像几乎没有什么变化，相反，经过腐蚀、膨胀处理后，白色孤立点噪声在腐蚀时被消除了。此时应注意：膨胀和腐蚀的顺序不同，处理结果也不相同。程序显示在 List 5.3 和 List 5.4 中。

(a) | (b)
(c) | (d)

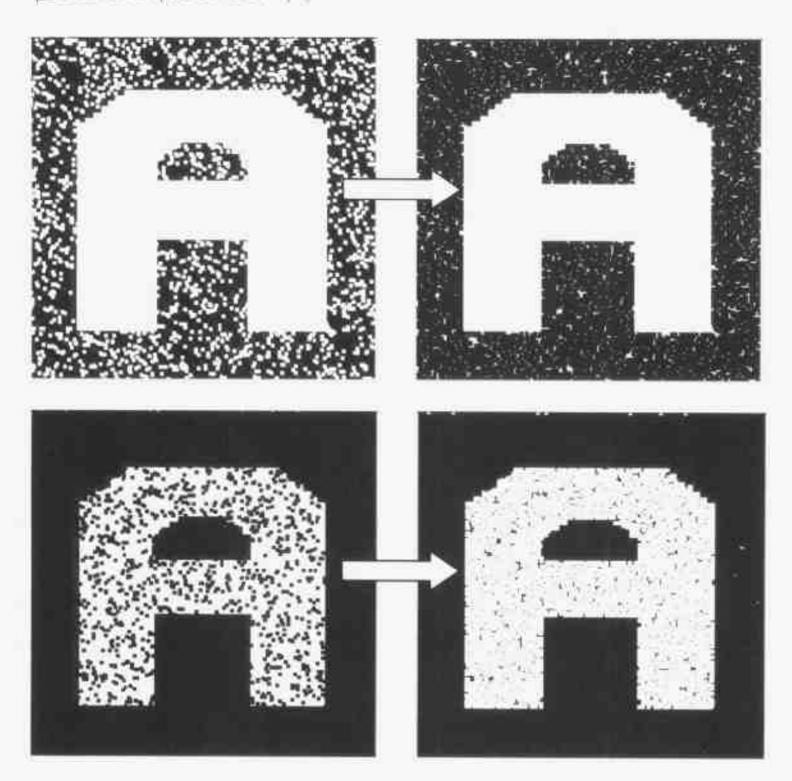


图 5-10

5.5 其他相关技术

前面所叙述的都是很平常但实际中使用最多的平滑方法。除此之外还有各种各样的方法，在此介绍一种稍微复杂的方法。

如图 5−11 所示，在某像素的周围 5×5 邻域内，依据 9 个模板选择 9 个像素。例如，在模板 1 中是 7 个像素，而在模板 9 中选择了 9 个像素。接着，对于各个模板求分散。

假设 n 个像素的值分别是 P_0、P_1、P_2、…P_{n-1}。



平均值（mean） $a=\frac{1}{n}\sum_{i=0}^{n-1}P_i$ (5.1)

分散（variance） $\sigma=\frac{1}{n}\sum_{i=0}^{n-1}(a-P_i)^2$ (5.2)

标准偏差（standard deviation） $d=\sqrt{\sigma}$ (5.3)

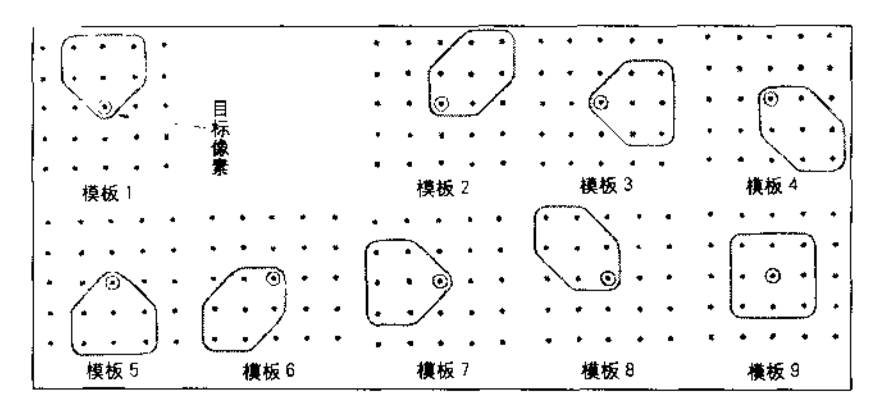


图 5-11

选择分散最小的模板，求其平均值作为目标像素的值。这实际上是把 5×5 邻域分割成图 5-12 所示的 9 个小区域，寻找其中最平坦的小区域（即无噪声和无边缘），然后输出，就可以消除噪声。

这种方法不仅在保持边缘的同时清除了噪声，而且有增强边缘的功能，因此被称为边缘保持平滑（edge preserving smoothing）。这种方法作为区域生成、边缘检测的预处理来进行是很有效的。但是计算时间也将会很长。

在消除噪声方面还没有很完善的方法，关键是要选择与之符合的方法。

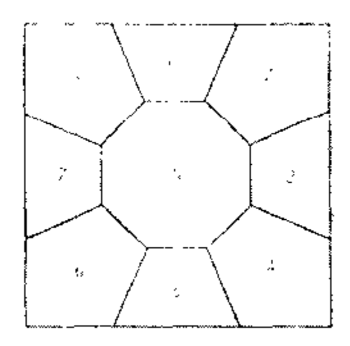


图 5-12



List 5.1　移动平均法

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
/*--- Image_smooth --- 去噪声处理（移动平均） ----------------------------
   image_in:  输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度

------------------------------------------------------------------------*/
void Image_smooth(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
   int            i, j, buf;

     for (j = 1; j < ysize-1; j++) {
     for (i = 1; i < xsize-1; i++) {
       buf = (int)(*(image_in + (j-1)*xsize + i-1))
          + (int)(*(image_in + (j-1)*xsize + i))
          + (int)(*(image_in + (j-1)*xsize + i+1))
          + (int)(*(image_in + j*xsize + i-1))
          + (int)(*(image_in + j*xsize + i))
          + (int)(*(image_in + j*xsize + i+1))
          + (int)(*(image_in + (j+1)*xsize + i-1))
          + (int)(*(image_in + (j+1)*xsize + i))
          + (int)(*(image_in + (j+1)*xsize + i+1));
       *(image_out + j*xsize + i) = (BYTE)(buf / 9);
     }
   }
}
```

List 5.2　中值滤波

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
/*--- Median --- 去噪声处理（中值） -------------------------------------
```



```
  image_in:    输入图像数据指针
  image_out:   输出图像数据指针
  xsize:       图像宽度
  ysize:       图像高度

-----------------------------------------------------------------------*/
int median_value(BYTE c[9]);

void Median(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
  int           i, j;
    unsigned char  c[9];

  for (i = 1; i < ysize-1; i++) {
    for (j = 1; j < xsize-1; j++) {
      c[0] = *(image_in + (i-1)*xsize + j-1);
      c[1] = *(image_in + (i-1)*xsize + j);
      c[2] = *(image_in + (i-1)*xsize + j+1);
      c[3] = *(image_in + i*xsize + j-1);
      c[4] = *(image_in + i*xsize + j);
      c[5] = *(image_in + i*xsize + j+1);
      c[6] = *(image_in + (i+1)*xsize + j-1);
      c[7] = *(image_in + (i+1)*xsize + j);
      c[8] = *(image_in + (i+1)*xsize + j+1);
      *(image_out + i*xsize + j) = median_value(c);
    }
  }
}

/*--- median_value ---求 9 个像素的中央值 ---------------
 c:  像素配列
-----------------------------------------------------------------------*/
int median_value(BYTE c[9])
{
  int    i, j, buf;

  for (j = 0; j < 8; j++) {
```



```
      for (i = 0; i < 8; i++) {
        if (c[i+1] < c[i]) {
          buf = c[i+1];
          c[i+1] = c[i];
          c[i] = buf;
        }
      }
    }
    return c[4];
}
```

List 5.3　腐蚀处理

```
#include "StdAfx.h"
#include "BaseList.h"
/*   Erodible   腐蚀 ------------------------------------------    ---
    image_in:      输入图像数据指针
    image_out:     输出图像数据指针
    xsize:         图像宽度
    ysize:         图像高度
-------------------------------------------------------------------*/
void Erodible(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
    int i, j;

    for (j = 1; j < ysize-1; j++) {
      for (i = 1; i < xsize-1; i++) {
        *(image_out + j*xsize + i) = *(image_in + j*xsize + i);
        if (*(image_in + (j-1)*xsize + i-1) == LOW)
          *(image_out + j*xsize + i) = LOW;
        if (*(image_in + (j-1)*xsize + i) == LOW)
          *(image_out + j*xsize + i) = LOW;
        if (*(image_in + (j-1)*xsize + i+1) == LOW)
          *(image_out + j*xsize + i) = LOW;
        if (*(image_in + j*xsize + i-1) == LOW)
          *(image_out + j*xsize + i) = LOW;
        if (*(image_in + j*xsize + i+1) == LOW)
```



```
        *(image_out + j*xsize + i) = LOW;
      if (*(image_in + (j+1)*xsize + i-1) == LOW)
        *(image_out + j*xsize + i) = LOW;
      if (*(image_in + (j+1)*xsize + i) == LOW)
        *(image_out + j*xsize + i) = LOW;
      if (*(image_in + (j+1)*xsize + i+1) == LOW)
        *(image_out + j*xsize + i) = LOW;
    }
  }
}
```

List 5.4 膨胀处理

```
#include "StdAfx.h"
#include "BaseList.h"
/*--  Dilation --- 膨胀 ----------------------------------------
   image_in:    输入图像数据指针
   image_out:   输出图像数据指针
   xsize:       图像宽度
   ysize:       图像高度
-------------------------------------------------------------*/
void Dilation(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
  int i, j;

  for (j = 1; j < ysize-1; j++) {
    for (i = 1; i < xsize-1; i++) {
      *(image_out + j*xsize + i) = *(image_in + j*xsize + i);
      if (*(image_in + (j-1)*xsize + i-1) == HIGH)
        *(image_out + j*xsize + i) = HIGH;
      if (*(image_in + (j-1)*xsize + i) == HIGH)
        *(image_out + j*xsize + i) = HIGH;
      if (*(image_in + (j-1)*xsize + i+1) == HIGH)
        *(image_out + j*xsize + i) = HIGH;
      if (*(image_in + j*xsize + i-1) == HIGH)
        *(image_out + j*xsize + i) = HIGH;
      if (*(image_in + j*xsize + i+1) == HIGH)
        *(image_out + j*xsize + i) = HIGH;
```



```
        if (*(image_in + (j+1)*xsize + i-1) == HIGH)
          *(image_out + j*xsize + i) = HIGH;
        if (*(image_in + (j+1)*xsize + i) == HIGH)
          *(image_out + j*xsize + i) = HIGH;
        if (*(image_in + (j+1)*xsize + i+1) == HIGH)
          *(image_out + j*xsize + i) = HIGH;
      }
    }
}
```





第6章

图像增强

6.1 清晰图像

图像是一种非常有用的信息源，所以要求它是清晰的。清晰图像是指对象物体的亮度和色彩的细微差别能清清楚楚地显示出来。清晰图像可以提供许多信息 可是通过摄像机所得到的图像并不一定是清晰的。例如，黑暗中拍摄的动物或者草丛中的蝗虫，目标物融入了具有相似亮度或者色彩的背景之中，这样的图像就难以分辨了。即使是这样的图像，对动物、蝗虫与背景之间在色彩和亮度上的差进行增幅，使背景中的动物和蝗虫的姿态显现出来，这也是可以做到的。像这样对图像中包含的亮度和色彩等信息进行增幅，或者将这些信息变换成其他形式的信息，然后通过各种手段来获得清晰图像的方法被称为图像增强（image enhancement）。

图像增强，其实就是给图像化妆。比如，为了使我们自己面部的眼睛和鼻子显得端庄漂亮，就会按照自己的方式对眼睛和鼻子进行化妆，加强眼睛和鼻子的效果。同样，图像增强是为了使图像清晰些所做的“化妆”。化妆中要用到口红、眼影、粉底等，根据加强部分的不同，需要用各种各样的化妆品，而图像的增强，因为增强的信息不同，有边缘增强、灰度增强、色彩的饱和度增强等方法。

有好几种方法可以用来增强图像，在此只对没有灰度差而难于分辨的图像进行处理，介绍一种使其变成清晰图像简单且有效的方法——对比度增强（contrast enhancement）。

6.2 对比度增强

画面明亮部分与阴暗部分灰度的比值称为对比度（contrast）。对比度高的图像中物体的轮廓分明，为清晰图像；相反地，对比度低的图像中物体轮廓模糊，为不清晰图像。例如，当看见一张很久以前留下的照片时，会发现它整个发白，并且黑白很难分辨清楚。对这种对比度低的图像，能够使其白的部分更白、黑的部分更黑，这种变换即对比度增强，从而得到清晰图像。

下面来说明对比度增强的方法。

如图 6–1 所示的图像，整个图像很暗，查看它的灰度直方图，如图 6–2 所示，会发现图像的灰度值都集中在灰度区域的低端。那么，如何对这样的图像进行处理使其变为清晰图像呢？可见，只要把过于集中的灰度值分散，使背景与对象物之间的灰度差扩大即可。一种处理方法是把图像中像素的灰度值都扩大 n 倍，即：

$$g(x,y)=n\times f(x,y) \tag{6.1}$$

在原始图像的位置（x,y）处的图像灰度值 $f(x,y)$ 被乘以 n，处理图像在（x,y）的灰度值就变为 $g(x,y)$。这个处理的程序被列在 List 6.1 中。因为图像灰度值范围是 0~255，所



以如果计算的结果超过 255，将其设定为 255，即把 255 作为限定的最大值。



图 6-1

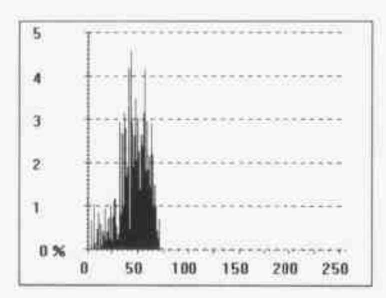


图 6-2

图 6-3 是经过图 6-1 的图像通过改变 n 值（1～6）的处理结果，其中，（a）是 n=1，（b）是 n=2，（c）是 n=3，（d）是 n=4，（e）是 n=5，（f）是 n=6。随着 n 值的增大，图像变得越来越亮，也越来越清晰。可是，当 n 值过大，图像整体变得白亮，反而难于分辨了。对这个图像来说，可以看出 n=3 时图像最为清晰，查看其灰度直方图（如图 6-4 所示）可知，当灰度值扩大 3 倍后，灰度分布几乎遍布 0～255 的整个区域，这样，图像的明暗分明，增强了其对比度。因此，可以顺次增加倍数 n 来寻求最佳值，以便得到清晰图像。那么，有没有通过对原始图像进行自动分析，实现自动增强对比度的方法呢？

(a)|(b)|(c)
(d)|(e)|(f)

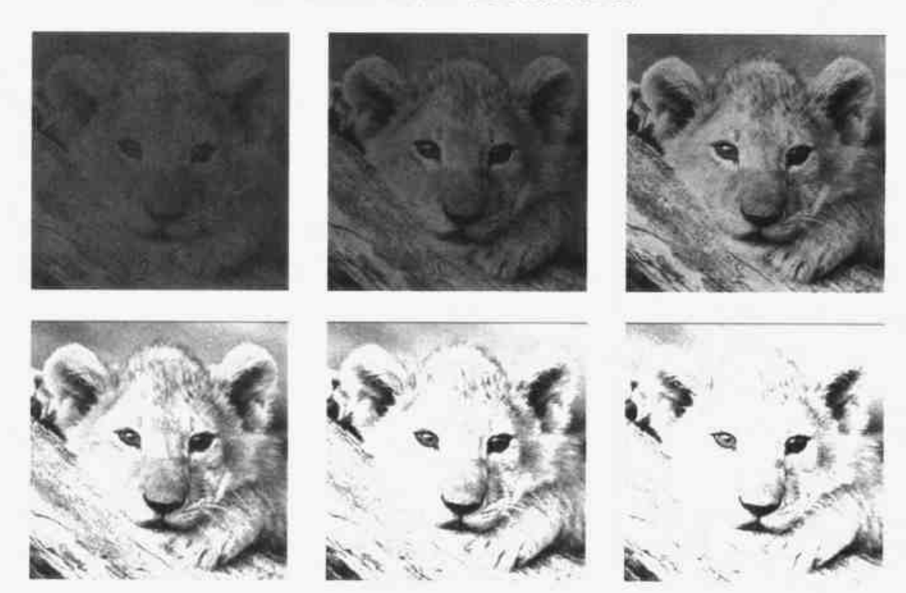


图 6-3



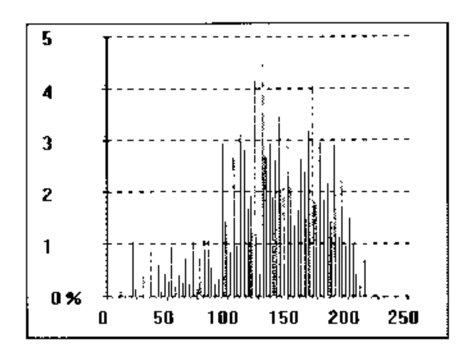


图 6-4

6.3 自动对比度增强

从上一节的结果可以看出，如果原始图像的灰度范围能够充满所允许的整个灰度范围，就可自动得到清晰图像。

对于灰度直方图，可以用式（6.2）将其范围从图 6–5 左侧所示的[a,b]变换到右侧所示的[a',b']。

$$z' = \frac{(b'-a')}{(b-a)} \times (z-a) + a' \tag{6.2}$$

根据这个式子就可以把任意像素的灰度 z（$a \leqslant z \leqslant b$）变换成灰度 z'。这个变换形式用灰度变换曲线来表现更易于理解。灰度变换曲线是用变换前图像的灰度值作为横坐标、变换后的灰度值作为纵坐标来表现的。式（6.2）的灰度变换曲线如图 6–6 所示。从这幅图可以看出，变换前的图像灰度的最小值 a 和最大值 b 分别被变换为 a'和 b'，任意值 z 被变换为 z'。那么，如果式（6.2）中的变量 a 和 b 是原始图像灰度值的最小值和最大值，变量 a'和 b'分别为内存所处理的灰度的最小值（0）和最大值（255），那么将自动得到从原始图像到对比度增强的图像。List 6.2 是求原始图像的灰度的最小值和最大值的程序，List 6.3 是通过式（6.2）进行变换的程序。图 6–7 为处理结果，图 6–8 为其灰度直方图，可见灰度值遍布了 0～255 的全部范围。图 6–7 与图 6–1 相比，对比度增强了，图像层次清晰分明。

然而，对于图 6–9 所示的原始图像，苹果和枝叶都比较暗。对比度增强后的结果如图 6–10 所示。这个结果与原始图像（图 6–9）相比，发现并没有变得清晰，查看其原始图像（图 6–9）的灰度直方图，如图 6–11 所示。在灰度直方图上，虽然中间的大部分区域像素点很少，但是低端和高端分别存在着像素数很多的灰度级，这样灰度直方图无法拉伸，当然也就无法进行对比度增强了。



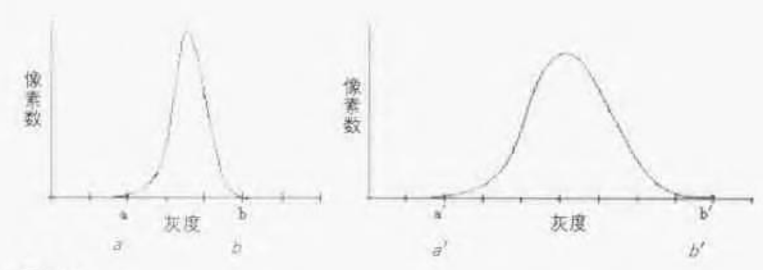


图 6-5

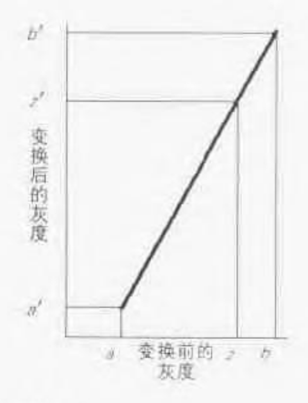


图 6-6



图 6-7

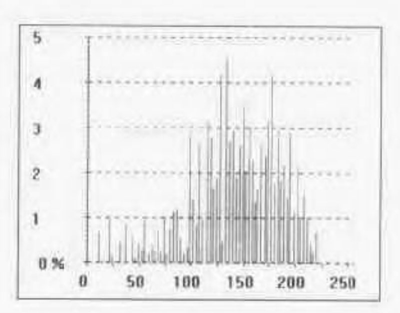


图 6-8





图 6-9



图 6-10

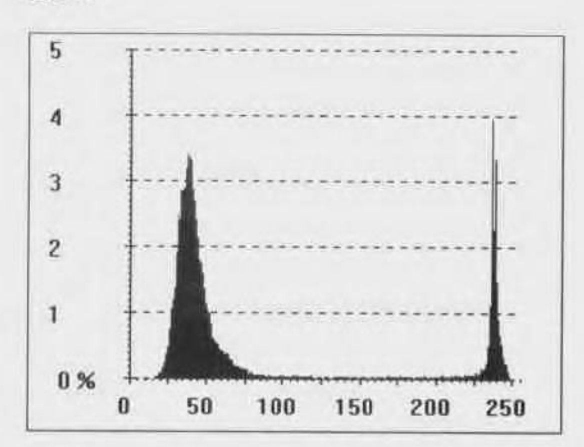


图 6-11

对于这种情况，图像对比度增强的方法有以下两种。



一种方法是将像素数少的灰度级压缩，只取出要增强部分的灰度值范围，进行灰度范围变换（gray-scale transformation 或 gray-level transformation）。也就是在式（6.2）中不把 a 和 b 作为灰度的最小值和最大值，而是把要增强的部分作为最小值和最大值，这样就可以直接使用 List 6.3 的程序了。

另一种方法是将灰度直方图上的所有灰度变换成像素数相同的分布形式。这种方法被称为灰度直方图均衡化（histogram equalization）。

前一种方法需要了解要增强部分的灰度范围，而后一种方法不需要查看灰度范围就可以进行对比度增强。下面对后一种方法进行比较详细的说明。

6.4 直方图均衡化

直方图均衡化（histogram equalization）是采取压缩原始图像中像素数较少的部分，拉伸像素数较多的部分的处理方法。如果在某一灰度范围内像素比较集中，因为被拉伸部分的像素相对于被压缩的部分要多，从而整个图像的对比度会增强，图像变得清晰。

下面用一个简单的例子来说明直方图均衡化算法。灰度为 0～7 的各个灰度级（gray level）所对应的像素数如图 6-12 所示。均衡化后，每个灰度级所分配的像素数，应该是总像素数除以总灰度级，即 40÷8=5。从原始图像灰度值大的像素开始，每次取 5 个像素，从 7 开始重新进行分配。对于图 6-12 所示的图像，给灰度级 7 分配原始图像中的灰度级 7、6 的全部像素，以及灰度级 5 的 9 个像素中的 1 个像素。从灰度级 5 的像素中选取 1 个像素的方法有如下两种。

- 随机选取。
- 从周围像素的平均灰度中的较大像素中顺次选取。

算法 2 比算法 1 稍微复杂一些，但是算法 2 所得结果的噪声比算法 1 少。

在此选用算法 2。使用前面的方法，从原始图像的灰度级 5 剩下的 8 个像素中选取 5 个，作为灰度级 6 的像素。以此类推，对所有像素重新进行灰度级分配。这个处理的程序表示在 List 6.4 中。在 List 6.4 中新的图像数组中按灰度级从大到小的顺序分配像素，最后的灰度级 0 是分配后剩下的全部像素。

灰度级	7	6	5	4	3	2	1	0
原始图像各灰度级的像素数	0	4	9	11	5	7	4	0
均衡化后各灰度级的像素数	5	5	5	5	5	5	5	5

图 6-12



图 6−13 和图 6−14 是利用这个程序分别对图 6−1 和图 6−9 进行直方图均衡化处理的结果。可见，两个例子都表明直方图均衡化对改善对比度是相当有效的。



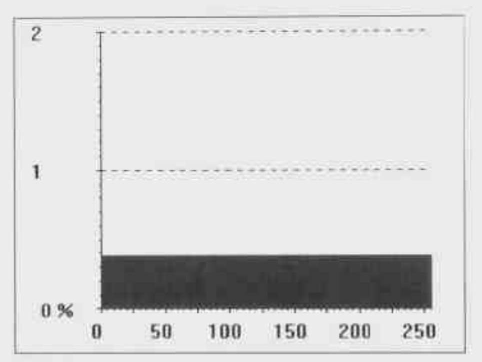


图 6-13



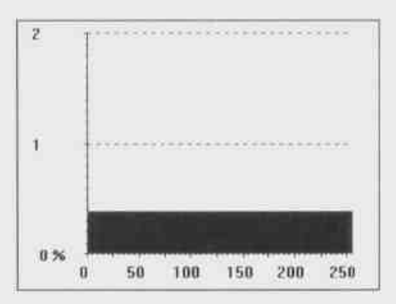


图 6-14

6.5 伪彩色增强

以上内容都是针对通过增加灰度变化来增强对比度，使图像层次清晰的方法。在这一节中，将介绍把图像中的灰度差用不同色彩来替代，从而使灰度图像通过灰度的细微差别，变换成易于区分的伪彩色图像（pseudo color image）。

使用伪彩色图像处理（pseudo color image processing）的一个例子就是测定人体温度分布，在人体图像上把所测定的结果从体温较低部位到体温较高的部位，赋予从蓝到红的不同颜色，这样就形成了一个能显示体温的伪彩色图像。人体图像首先把温度的高低变换成图像的明暗，也就是作为所谓的灰度图像被拍摄。然后利用伪彩色增强（pseudo color enhancement）来为灰度图像赋色。图 6−15 表示了一幅人体腿部的红外线图像，红色部分表示有软组织损伤的部位。



观看处理后的图像会发现体温随着颜色的变化而变化，另外根据红色表示温度高、蓝色表示温度低的要求，体温的高低会一目了然。

此外，对于其他无法得到彩色照片的情况，例如，人造卫星的红外线摄影、X 线拍照等，对拍摄到的照片赋予伪彩色，同样可以增强其辨别能力。

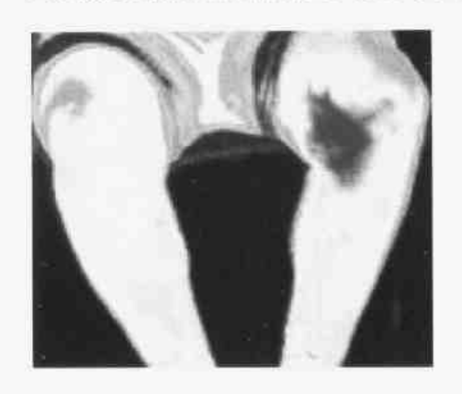


图 6-15

List 6.1　灰度n倍

```
#include "StdAfx.h"
#include "BaseList.h"
/*---Brightness_amplify --- 灰度 n 倍 ---------------------------------
   image_in:   输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   n:          倍数
-------------------------------------------------------------*/
void Brightness_amplify(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float n)
{
   int i, j, nf;

   for (j = 0; j < ysize; j++){
       for (i = 0; i < xsize; i++){
       nf = (int)(*(image_in + j*xsize + i) * n);
       if (nf > 255) nf = 255;
       *(image_out + j*xsize + i) = (BYTE)nf;
     }
```




```
    }
}
```

List 6.2 求取灰度范围

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Brightness_range --- 求取灰度范围 ------------------------------
   image_in:   输入图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   fmax:       求得的亮度最人值
   fmin:       求得的亮度最小值
-----------------------------------------------------------------*/
void Brightness_range(BYTE*image_in,int xsize,int ysize,int *fmax,int *fmin)
{
   int i, j, nf;

   *fmax = 0;
   *fmin = 255;
   for (j = 0; j < ysize; j++){
     for (i = 0; i < xsize; i++){
       nf =(int)(*(image_in + j*xsize + i));
       if (nf > *fmax) *fmax = nf;
       if (nf < *fmin) *fmin = nf;
     }
   }
}
```

List 6.3 灰度拉伸

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Brightness_expand --- 灰度范围拉伸 --------------------------
   image_in:   输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
```



```
    fmax:      输入图像亮度最大值
    fmin:      输入图像亮度最小值
------------------------------------------------------------------ */
void Brightness_expand(BYTE*image_in,BYTE*image_out,int xsize,int ysize,
int fmax, int fmin)
{
    int i, j;
    float d;

    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        d = (float)255 / (float)(fmax - fmin)
          * ((int)(*(image_in + j*xsize + i)) - fmin);
        if (d > 255)
          *(image_out + j*xsize + i) = 255;
        else if (d < 0)
          *(image_out + j*xsize + i) = 0;
        else
          *(image_out + j*xsize + i) = (BYTE)d;
      }
    }
}
```

List 6.4　直方图均衡化

```
#include "StdAfx.h"
#include <math.h>
#include "BaseList.h"
void sort(BYTE *image_in, int xsize, int ysize, struct XYW *data,
 int level);
void weight(BYTE *image_in, int xsize, int ysize, int i, int j,int *wt);

/*    Hist_plane --- 直方图均衡化 -----------------------------------
    image_in: 输入图像数据指针
    image_out:  输出图像数据指针
    xsize:      图像宽度
    ysize:      图像高度
    hist:       亮度直方图配列
```



```
---------------------------------------------------------------*/
void Hist_plane(BYTE*image_in,BYTE*image_out,int xsize,int ysize,long hist[256])
{
    int i, j, jy, ix, sum;
    int delt;               //根据周围像素值选择的像素数
    int low, high;          //处理灰度范围
    int av;                 //均衡化后的亮度值
    BYTE *image_buf;        //作业图像数据指针
    int max_number;
    XYW *buf;          //像素位置(x,y)及周边像素的亮度和(weight)

    if( (image_buf = new BYTE[xsize*ysize]) == NULL ) return;

    max_number = 0;
    for(i=0; i<256; i++) max_number = max(hist[i], max_number);

    if( (buf = new XYW[max_number]) == NULL ) return;

    av = (int)((ysize) * (xsize) / 256);
    high = 255;
    low = 255;
    for (j = 0; j < ysize; j++){
      for (i = 0; i < xsize; i++){
        *(image_out + j*xsize + i) = 0;
        *(image_buf + j*xsize + i) = *(image_in + j*xsize + i);
      }
    }

    for (i = 255; i > 0; i--){
      for (sum = 0; sum < av; low--) sum = sum + hist[low];
      low++;
      delt = hist[low] - (sum - av);
      sort(image_buf, xsize, ysize, buf, low);
      if (low < high){
        for (jy = 0; jy < ysize; jy++){
          for (ix = 0; ix < xsize; ix++){
            if (((int)(*(image_buf + jy*xsize + ix)) >= low + 1 ) &&
```



```
            ((int)(*(image_buf + jy*xsize + ix)) <= high))
          *(image_out + jy*xsize + ix) = (BYTE)i;
        }
      }
    }
    for (j = 0; j < delt; j++){
      *(image_out + buf[j].y*xsize + buf[j].x) =(BYTE)i;
      *(image_buf + buf[j].y*xsize + buf[j].x) =(BYTE)0;
    }
    hist[low] = hist[low] - delt;
    high = low;
  }

  delete image_buf;
  delete buf;
}

/*--- sort --- 将周围像素按灰度由高到低排列 ---------------------
  image_in:    输入图像数据指针
  xsize:       图像宽度
  ysize:       图像高度
  data:        位置及周围像素灰度和配列
  level:       排序像素的灰度
-----------------------------------------------*/
void sort(BYTE *image_in, int xsize, int ysize, struct XYW *data, int level)
{
  int i, j, inum, wt;
  struct XYW temp;

  inum = 0;
  for (j = 0; j < ysize; j++ ){
    for (i = 0; i < xsize; i++ ){
      if ((int)(*(image_in + j*xsize + i)) == level){
        weight(image_in,xsize,ysize,i,j,&wt); //计算周围像素的亮度和
        data[inum].y = j;
        data[inum].x = i;
        data[inum].w = wt;
```



```
          inum++;
        }
      }
    }

    for (j = 0; j < inum - 1; j++ ){          //排序
      for (i = j + 1; i < inum; i++){
        if (data[j].w <= data[i].w){
          temp.y = data[j].y;
          temp.x = data[j].x;
          temp.w = data[j].w;
          data[j].y = data[i].y;
          data[j].x = data[i].x;
          data[j].w = data[i].w;
          data[i].y = temp.y;
          data[i].x = temp.x;
          data[i].w = temp.w;
        }
      }
    }
}

/*--- weight ---计算周围像素的灰度和 ---------------------------------
    image_in:    输入图像数据指针
    xsize:       图像宽度
    ysize:       图像高度
    i, j:        像素位置
    wt:          灰度和
-------------------------------------------------------------------*/
void weight(BYTE *image_in, int xsize, int ysize, int i, int j, int *wt)
{
    int dim, djm;
    int dip, djp;
    int k, d[8];

    dim = i - 1;
    djm = j - 1;
```



```
  dip = i + 1;
  djp = j + 1;
  if (dim < 0) dim = i;
  if (djm < 0) djm = j;
  if (dip > xsize-1) dip = i;
  if (djp > ysize-1) djp = j;

  d[0] = (int)(*(image_in + djm*xsize + dim));
  d[1] = (int)(*(image_in + djm*xsize + i));
  d[2] = (int)(*(image_in + djm*xsize + dip));
  d[3] = (int)(*(image_in + j*xsize + dim));
  d[4] = (int)(*(image_in + j*xsize + dip));
  d[5] = (int)(*(image_in + djp*xsize + dim));
  d[6] = (int)(*(image_in + djp*xsize + i));
  d[7] = (int)(*(image_in + djp*xsize + dip));

  *wt = 0;
  for (k = 0; k < 8; k++) *wt = *wt + d[i];
}
```





第7章

特征选择与描述

7.1 基于图像特征的自动识别

有时，一个人所持有钥匙的多少成了他地位的象征，假如你除了自家的门钥匙、自己办公室的门钥匙、自家的车钥匙之外，还有许多其他的门钥匙，甚至还持有金库的钥匙，你一定是个大权在握的人。

但是随着图像处理技术的发展，也许将来就没有了钥匙。现在由于计算机技术的进步，有许多钥匙已经变成了电子键、电子暗号等。如果有一天，计算机能够判别人的脸部特征或者声音特征，那么就没有必要记暗号或者持钥匙了。目前，通过计算机识别图像特征，对物体进行自动判别的例子已经很多。例如自动售货机的钱币判别、工厂内通过摄像机自动判别产品质量、通过判别邮政编码自动分检信件、基于指纹识别的电子钥匙以及最近出现的通过脸型识别来防范恐怖分子等。本章就对这些特征（feature），尤其是图像的特征选择（feature selection）进行说明。

为了便于理解，本章以简单的二值图像为对象，通过识别物体的形状、大小等特征，介绍提取所需要的物体、除去噪声的方法。

7.2 二值图像的特征参数

所谓图像的特征（feature），就是指图像中包括具有某种特征的物体。如图 7−1 所示，在图像上有一些水果，如果希望从该图像中提取出香蕉，对于计算机来说，它并不知道人们讲的香蕉是什么，只能通过提供所要提取物体的特征来指示计算机。例如，香蕉是细长的物体，那么可以通过这个特征来指示计算机。也就是说，必须告诉计算机图像中物体的大小、形状等，指出该物体的特征。当然，这种指示依靠的是描述物体形状特征（shape representation and description）的参数。

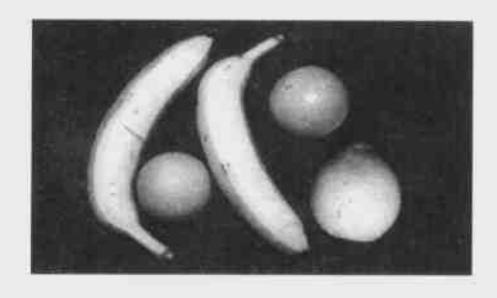


图 7-1

下面说明几个有代表性的特征参数以及计算方法。表 7–1 列出了几个图形以及相应的参数。

表 7-1　图形及其特征

种　类	圆	正方形	正三角形
图像			
面积	πr^2	r^2	$\frac{\sqrt{3}}{4}r^2$
周长	$2\pi r$	$4r$	$3r$
圆形度	1.0	$\frac{\pi}{4}=0.79$	$\frac{\pi\sqrt{3}}{9}=0.60$

1 / 面积（area）

计算物体（或区域）中包含的像素数。

2 / 周长（perimeter）

物体（或区域）轮廓线的周长是指轮廓线上像素间距离之和。像素间距离有图 7–2（a）和（b）两种情况。图 7–2（a）表示并列的像素，当然并列方式可以是上、下、左、右四个方向，这种并列像素间的距离是 1 个像素。图 7–2（b）表示的是倾斜方向连接的像素，倾斜方向也有左上角、左下角、右上角、右下角四个方向，这种倾斜方向像素间的距离是 $\sqrt{2}$ 像素。在进行周长测量时，需要根据像素间的连接方式，分别计算距离。图 7–2（c）是一个周长的测量示例。

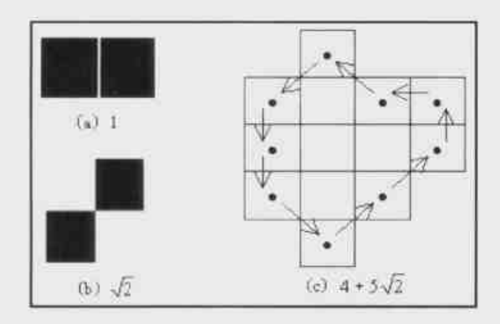


图 7-2

如图 7–3 所示，提取轮廓线，需要按以下步骤对轮廓线进行追踪。



⑴ 扫描图像　顺序调查图像上各个像素的值，寻找没有扫描标志 a_0 的边界点。

⑵ 如果 a_0 周围全为黑像素（0），说明 a_0 是个孤立点，停止追踪。

⑶ 否则，按图 7–3 的顺序寻找下一个边界点。用同样的方法，追踪其他的边界点。

⑷ 到了下一个交界点 a_0，证明已经围绕物体一周，终止扫描。

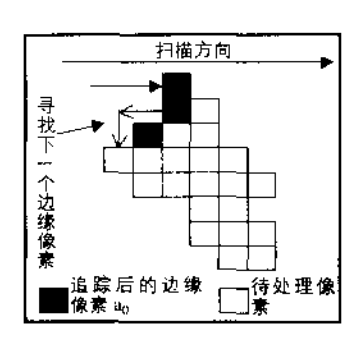


图 7-3

3 / 圆形度（compactness）

圆形度是在面积和周长的基础上，计算物体（或区域）的形状复杂程度的特征量。例如，可以考察一下圆和五角星。如果五角星的面积和圆的面积相等，那么它的周长一定比圆长。因此，可以考虑以下参数：

$$e=\frac{4\pi\times\text{面积}}{(\text{周长})^2} \tag{7.1}$$

e 就是圆形度。对于半径为 r 的圆来说，面积等于 πr^2，周长等于 $2\pi r$，所以圆形度 e 等于 1。由表 7–1 可以看出，形状越接近于圆，e 越大，最大为 1，形状越复杂 e 越小，e 的值在 0 和 1 之间。

4 / 重心（center of gravity或centroid）

重心就是求物体（或区域）中像素坐标的平均值。例如，某白色像素的坐标为（x_i，y_i）（i ＝ 0，1，2，…，n–1），其重心坐标（x_0，y_0）可由下式求得：

$$(x_0,y_0)=\left(\frac{1}{n}\sum_{i=0}^{n-1}x_i,\frac{1}{n}\sum_{i=0}^{n-1}y_i\right) \tag{7.2}$$

除了上面的参数以外，还有长度和宽度（length and breadth）、欧拉数（Euler's number）以及可查看物体长度方向的区域矩（moment）等许多特征参数，这里就不一一介绍了。

利用上述参数，只能把香蕉与其他水果区别开。香蕉是那些水果中圆形度最小的。不过，首先需要把所有的东西从背景中提取出来，这可以利用二值化处理提取明亮部分来完成。图 7–4 是图 7–1 的图像经过二值化处理（阈值为 40 以上），再通过两次中值滤波去噪声处理后



的图像。

到此为止，还必须将每一个物体区分开。为了区分每个物体，必须调查像素是否连接在一起，这样的处理称为区域标记（label）。

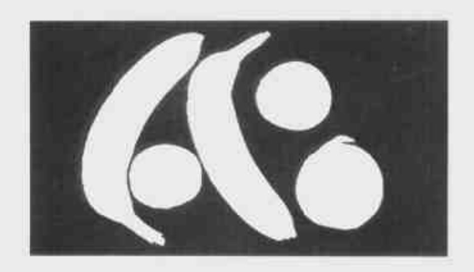


图 7-4

7.3　区域标记

区域标记是指给连接在一起的像素（称为 connected component，连接成分）附上相同的标记，不同的连接成分附上不同的标记。区域标记在二值图像处理中占有非常重要的地位。图 7-5 表示了区域标记后的图像，通过该处理将各个连接成分区分开，然后就可以调查各个连接成分的形状特征了。

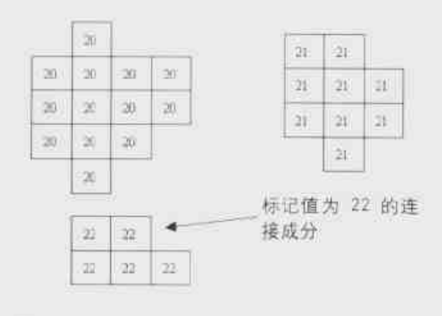


图 7-5

区域标记也有许多方法，下面介绍一种简单的方法。步骤如下（参考图 7-6）：



1 扫描图像，遇到没加标记的目标像素（白像素）P 时，添加一个新的标记（label）。

2 给与 P 连接在一起（即相同连接成分）的像素添加相同的标记。

3 进一步给所有与加标记像素连接在一起的像素添加相同的标记。

4 直到连接在一起的像素全部被添加标记。这样，一个连接成分就被添加了相同的标记。



⑤ 返回到第①步，重新查找新的没加标记的像素，重复上述各个步骤。

⑥ 图像全部被扫描后，处理结束。

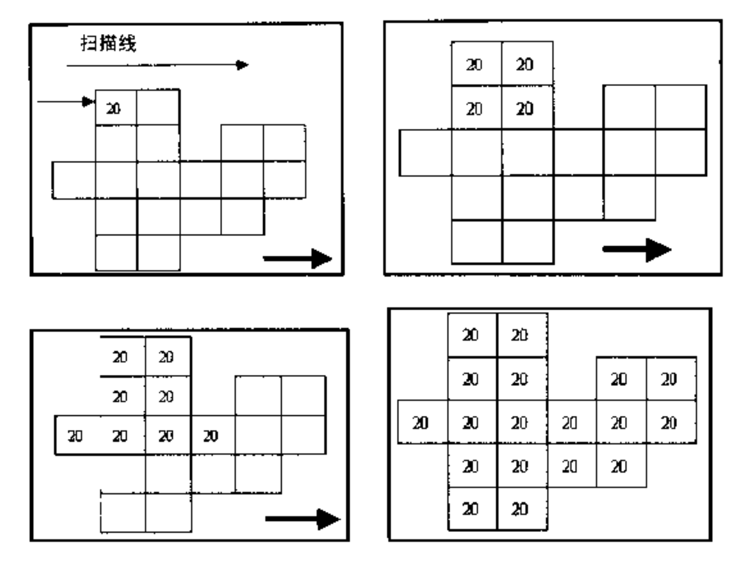


图 7-6

区域标记处理的程序显示在 List 7.1 中，识别物体（连接成分）特征参数的处理程序显示在 List 7.2 中。

7.4 基于特征参数提取物体

通过以上处理，完成了从图 7–1 中提取香蕉的准备工作。识别各个物体特征的步骤如图 7–7 所示，处理结果显示在表 7–2 中。图 7–8 所示为处理后的图像，轮廓线和重心位置的像素显得比较亮。

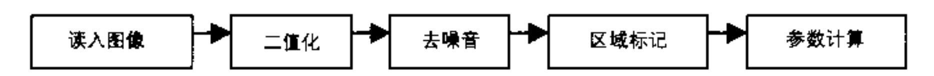


图 7-7

由表 7–2 可知，圆形度小的物体有两个，这两个可能就是香蕉。如果要提取香蕉，按照图 7–7 的步骤进行处理，然后再把具有某种圆形度的连接成份提取即可。上述处理的程序显示在 List 7.3 中，该程序中把圆形度小于最小 ratio_min 或者大于最大 ratio_max 的连接成分的像素



值设置为0。提取的连接成分的图像如图7-9所示。这些处理获得了一个掩模图像(mask image)，利用该掩模即可从原始图像（图7-1）中把香蕉提取出来。提取结果如图7-10所示。用掩模图像进行提取的程序显示在List 7.4中。

表7-2　各个物体的特征参数

物体序号	面积（像素）	周长（像素）	圆形度	重心位置（像素）
0	21718	894.63	0.3410	(307, 209)
1	22308	928.82	0.3249	(154, 188)
2	9460	367.85	0.8785	(401, 136)
3	14152	495.14	0.7454	(470, 274)
4	8570	352.98	0.8644	(206, 260)

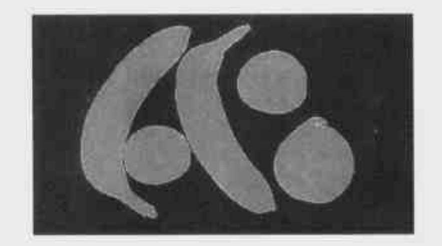


图7-8

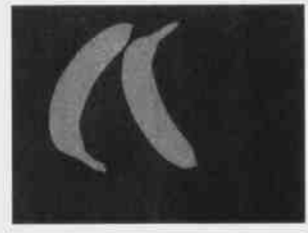


图7-9

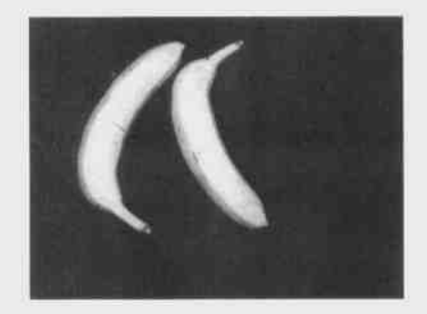


图7-10

7.5　基于特征参数消除噪声

到现在为止，都是以提取物体为目标在进行处理，当然也可以用除去不需要的对象的方法。例如，可以消去噪声。关于二值图像的噪声，在第5章介绍了由膨胀与腐蚀连接成分消除噪声的方法。但是利用特征参数也可以进行这个处理。也就是说，通过区域标记处理将各个连接成



分区分开后，除去面积小的连接成分即可。处理流程如图 7-11 所示，处理程序显示在 List 7.5 中，处理结果如图 7-12 所示，其中，（a）是原始图像，（b）是微分图像，（c）是二值图像（阈值 50），（d）是小于 10 像素的面积除去后的图像。将由微分处理（Prewitt 算子）所获得的图像（如图 7-12（c）所示）作为输入图像，消噪声处理后的结果图像如图 7-12（d）所示，被除去的噪声是面积小于 10 像素的连接成分，可见图中点状噪声完全消失了。

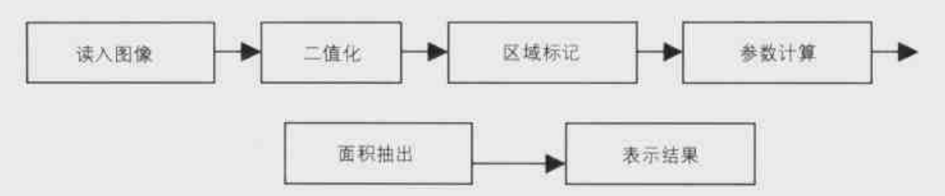


图 7-11

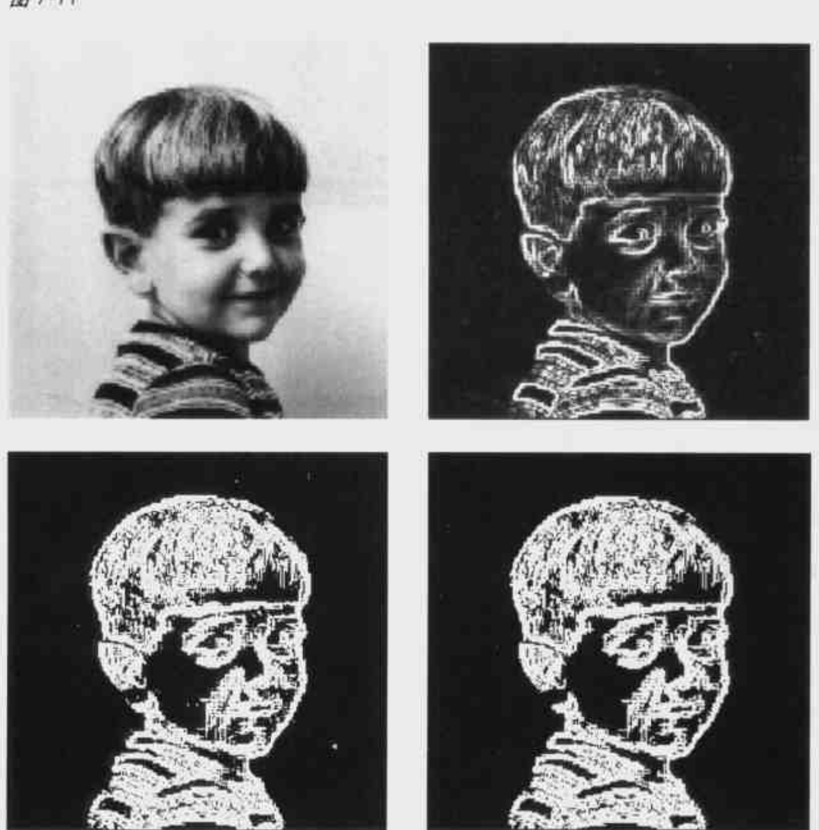


图 7-12



7.6 高级特征参数

物体最典型的特征是大小和圆形度。只要利用这两个简单的特征，就可以进行各种各样的处理。在实际应用中，这些参数的应用也很广泛，例如工厂内不合格品的检出，摄像机方向的控制、机器人手臂位置的控制等。

如果使用更高级的特征参数，可以进行更为复杂的同类判别。可应用数学式子来表示物体的边缘线，通过细线化处理求图形的骨骼等。关于细线化处理，在第 4 章的提取边缘中介绍过，如图 7-13 所示，可以用细线化处理求出图形的中心线，以便查看复杂图形的几何学特征。因为细线化处理以后，可以查出图形的结合情况，所以细线化图像可以用于文字、画面的解析。如果了解了图形的几何学特征，就可以进行邮政编码的自动判别、手写画面的自动识别等操作了。

一般的图像都具有浓淡层次的灰度特征和颜色特征，如果巧妙地利用这些特征，则可以进行更为高级的图像处理。例如，在人造卫星进行土地调查的图像处理中，利用灰度和颜色的变化情况，制作土地利用分布图等。



图 7-13

List 7.1　区域标记（加标记）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>

void labelset(BYTE *image, int xsize, int ysize, int xs, int ys, int label);

/*--- Labeling --- 加标记处理 ------------------------------------
   image_in:     输入图像数据指针（二值图像）
   image_out:    输出图像数据指针（标记图像）
```



```
   xsize:      图像宽度
   ysize:      图像高度
   cnt:        标记个数
-------------------------------------------------------------*/
int Labeling(BYTE*image_in, BYTE *image_out, int xsize, int ysize,int *cnt)
{
   int i, j, label;

   for (j = 0; j < ysize; j++)
     for (i = 0; i < xsize; i++)
       *(image_out + j*xsize + i) = *(image_in + j*xsize + i);
   label = L_BASE;
   for (j = 0; j < ysize; j++)
     for (i = 0; i < xsize; i++) {
       if (*(image_out + j*xsize + i) == HIGH) {
         if (label >= HIGH) {
           AfxMessageBox("Error! too many labels.");
           return -1;
           }
         labelset(image_out, xsize, ysize, i, j, label);
         label++;
       }
      }
   *cnt = label-L_BASE;

   return 0;
}

/*--- labelset --- 给连接像素加标记 --------------------
   image:      图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   xs, ys:     开始位置
   label:      标记值
-------------------------------------------------------------*/
void labelset(BYTE *image, int xsize, int ysize, int xs, int ys, int label)
{
```



```
int i, j, cnt, im, ip, jm, jp;

*(image + ys*xsize + xs) = label;
for (;;) {
  cnt = 0;
  for (j = 0; j < ysize; j++)
    for (i = 0; i < xsize; i++)
      if (*(image + j*xsize + i) == label) {
        im = i-1; ip = i+1; jm = j-1; jp = j+1;
        if (im < 0) im = 0; if (ip >= xsize) ip = xsize-1;
        if (jm < 0) jm = 0; if (jp >= ysize) jp = ysize-1;
        if (*(image + jm*xsize + im) == HIGH) {
          *(image + jm*xsize + im) = label; cnt++;
        }
        if (*(image + jm*xsize + i ) == HIGH) {
          *(image + jm*xsize + i ) = label; cnt++;
        }
        if (*(image + jm*xsize + ip) == HIGH) {
          *(image + jm*xsize + ip) = label; cnt++;
        }
        if (*(image + j*xsize + im) == HIGH) {
          *(image + j*xsize + im) = label; cnt++;
        }
        if (*(image + j*xsize + ip) == HIGH) {
          *(image + j*xsize + ip) = label; cnt++;
        }
        if (*(image + jp*xsize + im) == HIGH) {
          *(image + jp*xsize + im) = label; cnt++;
        }
        if (*(image + jp*xsize + i ) == HIGH) {
          *(image + jp*xsize + i ) = label; cnt++;
        }
        if (*(image + jp*xsize + ip) == HIGH) {
          *(image + jp*xsize + ip) = label; cnt++;
        }
      }
  if (cnt == 0) break;
```



```
    }
}
```

List 7.2 计算图像特征参数

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>

float calc_size(BYTE *image_label, int xsize, int ysize,
int label, int *cx, int *cy);
float calc_length(BYTE *image_label, int xsize, int ysize, int label);
float trace(BYTE *image_label, int xsize, int ysize, int xs, int ys);

/*--- Features  -- 计算特征数据 ------------------------------------
   image_label_in:   输入标记图像指针
   image_label_out:  输出标记图像指针
   xsize:            图像宽度
   ysize:            图像高度
   cnt:              对象物个数
   size:             面积
   length:           周长
   ratio:            圆形度
   center_x:         重心 x 坐标
   center_y:         重心 y 坐标

------------------------------------------------------------------*/
void Features(BYTE*image_label_in,BYTE*image_label_out,int xsize,int ysize,
int cnt,float size[],float length[],float ratio[],int center_x[],int center_y[])
{
   int   i, j, cx, cy;
   float L;

   for (j = 0; j < ysize; j++){
     for (i = 0; i < xsize; i++){
       *(image_label_out + j*xsize + i)=*(image_label_in +j*xsize + i);
     }
```



```
    }

    for (i = 0; i < cnt; i++) {

      size[i] = calc_size(image_label_out, xsize, ysize, i+L_BASE,
        &cx, &cy);
      center_x[i] = cx;
      center_y[i] = cy;

      L = calc_length(image_label_out, xsize, ysize, i+L_BASE);
      length[i] = L;

      ratio[i] = 4*PI*size[i]/(L*L);
      *(image_label_out + cy*xsize + cx) = HIGH;  //重心

    }
}

/*--- calc_size --- 求面积和重心位置 -----------------------------------
    image_label:  标记图像指针
    xsize:        图像宽度
    ysize:        图像高度
    label:        标记号
    cx, cy:       重心位置
-------------------------------------------------------------------*/
float calc_size(BYTE *image_label, int xsize, int ysize,
 int label, int *cx, int *cy)
{
    int   i, j;
    float tx, ty, total;

    tx = 0; ty = 0; total = 0;
    for (j = 0; j < ysize; j++)
      for (i = 0; i < xsize; i++)
        if (*(image_label + j*xsize + i) == label) {
          tx += i; ty += j; total++;
        }
```



```
    if (total == 0.0) return 0.0;
    *cx = (int)(tx/total); *cy = (int)(ty/total);
    return total;
}

/*--- calc length --- 求周长 -----------------------------------------
    image_label:  标记图像指针
    xsize:        图像宽度
    ysize:        图像高度
    label:        标记号
-------------------------------------------------------------------*/
float calc_length(BYTE *image_label, int xsize, int ysize, int label)
{
    int   i, j;
    float leng = 1;

    for (j = 0; j < ysize; j++){
      for (i = 0; i < xsize; i++){
        if (*(image_label + j*xsize + i) == label)
        {
           leng = trace(image_label, xsize, ysize, i-1, j);
          return leng;
        }
      }
    }
    return 0;
}

/*--- trace --- 追踪轮廓线 -------------------------------------------
    image_label:  标记图像指针
    xsize:        图像宽度
    ysize:        图像高度
    xs, ys:       开始位置
-------------------------------------------------------------------*/
float trace(BYTE *image_label, int xsize, int ysize, int xs, int ys)
{
    int   x, y, no, vec;
```



```
float l;

l = 0;  x = xs; y = ys; no = *(image_label + y*xsize + x+1); vec = 5;
for (;;) {
  if (x == xs && y == ys && l != 0) return l;

  *(image_label + y*xsize + x) = HIGH;


  switch (vec) {
    case 3:
      if (*(image_label + y*xsize + x+1) != no &&
        *(image_label + (y-1)*xsize + x+1) == no)
        {x = x+1; y = y  ; l++        ; vec = 0; continue;}
    case 4:
      if (*(image_label + (y-1)*xsize + x+1) != no &&
        *(image_label + (y-1)*xsize + x) == no)
        {x = x+1; y = y-1; l += ROOT2; vec = 1; continue;}
    case 5:
      if (*(image_label + (y-1)*xsize + x) != no &&
        *(image_label + (y-1)*xsize + x-1) == no)
        {x = x  ; y = y-1; l++        ; vec = 2; continue;}
    case 6:
      if (*(image_label + (y-1)*xsize + x-1) != no &&
        *(image_label + y*xsize + x-1) == no)
        {x = x-1; y = y-1; l += ROOT2; vec = 3; continue;}
    case 7:
      if (*(image_label + y*xsize + x-1) != no &&
        *(image_label + (y+1)*xsize + x-1) == no)
        {x = x-1; y = y  ; l++        ; vec = 4; continue;}
    case 0:
      if (*(image_label + (y+1)*xsize + x-1) != no &&
        *(image_label + (y+1)*xsize + x) == no)
        {x = x-1; y = y+1; l += ROOT2; vec = 5; continue;}
    case 1:
      if (*(image_label + (y+1)*xsize + x) != no &&
        *(image_label + (y+1)*xsize + x+1) == no)
```



```
            {x = x  ; y = y+1; l++      ; vec = 6; continue;}
        case 2:
          if (*(image_label + (y+1)*xsize + x+1) != no &&
            *(image_label + y*xsize + x+1) == no)
            {x = x+1; y = y+1; l += ROOT2; vec = 7; continue;}
          vec = 3;
      }
    }
}
```

List 7.3 根据圆形度抽出物体

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>

/*--- Ratio_extract --- 抽出具有某圆形度的对象物 ------------------
   image_label_in:         输入标记图像指针
   image_label_out:        输出标记图像指针
   xsize:                  图像宽度
   ysize:                  图像高度
   cnt:                    对象物个数
   ratio:                  圆形度
   ratio_min, ratio_max: 最小值，最大值
----------------------------------------------------------------*/
void Ratio_extract(BYTE *image_label_in, BYTE *image_label_out, int xsize, int
ysize, int cnt, float ratio[], float ratio_min, float ratio_max)
{
   int i, j, x, y;
   int lno[256];

   for (i = 0, j = 0; i < cnt; i++)
   {
       if (ratio[i] >= ratio_min && ratio[i] <= ratio_max)
       lno[j++] = L_BASE+i;
   }
   for (y = 0 ; y < ysize; y++) {
```



```
      for (x = 0; x < xsize; x++) {
        *(image_label_out + y*xsize + x) = 0;
        for (i = 0; i < j; i++)
        {
            if (*(image_label_in + y*xsize + x) == lno[i])
             *(image_label_out+y*xsize+x)=*(image_label_in+y*xsize+x);
        }
      }
    }
}
```

List 7.4 复制掩模领域的原始图像

```
#include "StdAfx.h"
#include "BaseList.h"

/*---- Mask_copy --- 复制掩模领域的原始图像 -----------------------
   image_in:   输入图像指针
   image_out:  输出图像指针
   image_mask: 输入模块图像（二值图像）
   xsize:      图像宽度
   ysize:      图像高度
--------------------------------------------------------------*/
void Mask_copy(BYTE *image_in, BYTE *image_out,
 BYTE *image_mask, int xsize, int ysize)
{
   int i, j;

   for (j = 0; j < ysize; j++) {
     for (i = 0; i < xsize; i++) {
       if (*(image_mask + j*xsize +i) != LOW)
         *(image_out + j*xsize + i) = *(image_in + j*xsize + i);
       else *(image_out + j*xsize + i) = 0;
     }
   }
}
```



List 7.5　根据面积提取对象物

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>

/*--- Size_extract --- 抽出某面积范围的对象物 ---------------------
   image_label_in:      输入标记图像指针
   image_label_out:     输出标记图像指针
   xsize:               图像宽度
   ysize:               图像高度
   cnt:                 对象物个数
   size:                面积
   size_min, size_max: 最小、最大值
-----------------------------------------------------------*/
void Size_extract(BYTE *image_label_in, BYTE *image_label_out, int xsize, int
ysize, int cnt, float size[], float size_min, float size_max)
{
   int i, j, x, y;
   int lno[256];

   for (i = 0, j = 0; i < cnt; i++)
     if(size[i]>=size_min && size[i]<=size_max) lno[j++]=L_BASE+i;
   for (y = 0; y < ysize; y++) {
      for (x = 0; x < xsize; x++) {
       *(image_label_out + y*xsize + x) = 0;
       for (i=0 ; i<j ; i++)
         if (*(image_label_in + y*xsize + x) == lno[i])
           *(image_label_out+y*xsize+x)=*(image_label_in+y*xsize+x);
     }
   }
}
```



第8章

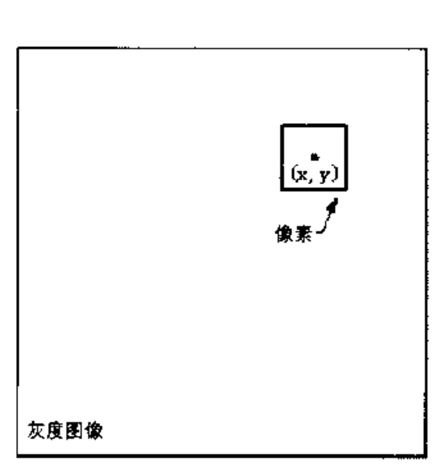
彩色变换

8.1 彩色信息处理

在彩色图像（color image）的处理过程中不仅要考虑位置、灰度，还要考虑彩色信息。位置信息在画面上是用指定的坐标系来表示的，灰度信息是用 0～255 的灰度级来处理的，那么彩色信息怎么样来取得呢？作为处理彩色图像的系统，最熟悉的莫过于彩色电视机了。通过彩色电视机每天都能欣赏到十分逼真的画面。远处观看所放映的图像时，能看到的是一个很平滑的影像，但当接近电视机的荧光屏，在近距离观看时，会发现影像是由一个个小点光源聚集起来所形成的。

这些点光源中包含着红（R）、绿（G）、蓝（B）等颜色。当绿草地被拍摄时，与 R、B 的点相比，G 点发出的光较强，可是对于蓝天的景色来说，与 R、G 相比 B 点发出的光较强。另外，对于其他部分，根据 R、G、B 的光强的不同可以显出各种各样的颜色。总之，各种强度的 R、G、B 光混合在一起就会产生出各种各样的颜色。

抛开图像处理的话题，我们知道彩色图像是由 R、G、B 三种颜色构成的，这三种颜色被称为视觉的三基色（three primary color）。在处理彩色图像的内存中，如图 8-1 右侧所示，为了记录 R、G、B 的颜色信息，需要备有分别储存它们的颜色平面。能够表示的颜色的数目是由 R、G、B 分别对应的颜色平面能够处理的灰度级数来决定的。当 R、G、B 的各个颜色平面用 1 像素 8 位的数据，即 256 级的灰度表示时，能够处理 256（R）×256（G）×256（B）=1677 万色的颜色。



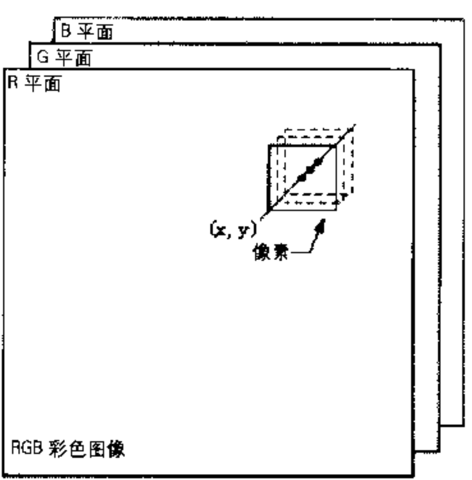


图 8-1



8.2 彩条制作

在了解了彩色图像在内存中的表示方式后，可以制作出彩色图像并在显示器上显示出来。调整摄像机和显示器时，经常在彩色图像上要用到彩条或色带（color bars）。如图 8–2 所示，彩条从左开始依次排列为白（white，W）、黄（yellow，Ye）、青（cyan，Cy）、绿（green，G）、品红（magenta，Mg）、红（red，R）、蓝（blue，B）、黑（black，Bl）。由于 W 是由 R、G、B 以同样的比例混合而成，Cy、Mg、Ye 分别是 G 和 B、R 和 B、R 和 G 混合而成，所以为了制作图 8–1 所示的彩条，在 R、G、B 的平面上写入如图 8–3 所示的数据就可以了，其中，白色表示明亮部分，灰色表示黑暗部分，（a）是 R 平面，（b）是 G 平面，（c）是 B 平面。制作彩条的程序列举在 List 8.1 中。所制作的彩条如图 8–4 所示。

W	Ye	Cy	G	Mg	R	B	Bl
白	黄	青	绿	品红	红	蓝	黑

图 8-2



图 8-3

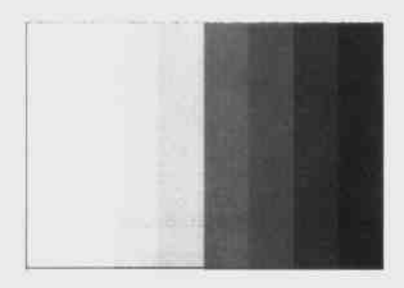


图 8-4



8.3 颜色描述

除了彩条所显示的颜色以外，还有许多各种各样的颜色。在表现这些颜色时，有浅红、略带蓝色的绿色等表达方式。在这种表达颜色的方式中，对于同一种颜色还有别的称呼方式，也许听到的是同样的颜色，可每个人脑子里所想的颜色却可能不相同，因此，为了尽可能定量地来表现颜色，可以把颜色分成如下三个特性来表示，第一个特性是色调或者色相（hue，H），可表现 *R*、*G*、*B*、*Ye* 等各种各样颜色的种类。第二个特性是用来表现明暗的，称为明度（value）或者亮度 Y（brightness，或 I（intensity））。最后，同样的颜色有浓淡之分，这就是用来表现颜色鲜明程度的饱和度或彩度 S（saturation）。这三个特性被称为颜色的三个基本属性。颜色的这三个基本属性可以用一个理想化的双锥体 HSI 模型来表示，图 8–5 显示了基于彩色三角形和圆形的双锥体 HSI 模型（HSI model based on color triangles and also on circles）。双锥体轴线代表非彩色系列（achromatic series），即与亮度坐标重合。垂直于轴线的平面可以用图 8–5 上面所示的彩色三角形来表示，也可以用图 8–5 下面所示的彩色圆来表示。色调和饱和度用极坐标形式表示，即夹角表示色调，径向距离表示在一定色调下的饱和度。

这三个基本属性之间的关系是三维的，从而能够改变在 *R*、*G*、*B* 平面上写入的数值，然后进行操作。可是如果还不清楚 *R*、*G*、*B* 的数值与色调、饱和度的关系，就很难使用这种方法进行操作。在此，首先考虑如何从 *R*、*G*、*B* 信号来分离亮度信号和彩色信号。

实际上，在制作彩色电视信号时，就是把 *R*、*G*、*B* 信号变换到亮度信号 *Y* 和颜色信号 C_1、C_2。其关系式如下

$$
\begin{aligned}
Y &= 0.3R + 0.59G + 0.11B \\
C_1 &= R - Y = 0.7R - 0.59G - 0.11B \\
C_2 &= B - Y = -0.3R - 0.59G + 0.89B
\end{aligned}
\tag{8.1}
$$

式（8.1）中表示了 *R*、*G*、*B* 信号与 *Y*、C_1、C_2 的关系。其中亮度信号 *Y* 相当于灰度图像，彩色信号 C_1、C_2 是除去了亮度信号所剩下的部分，称为色差信号（chrominance）。

相反，从亮度信号、色差信号求 *R*、*G*、*B* 的式子如下所示。

$$
\begin{aligned}
R &= Y + C_1 \\
G &= Y - \frac{0.3}{0.9}C_1 - \frac{0.11}{0.59}C_2 \\
B &= Y + C_2
\end{aligned}
\tag{8.2}
$$

从 *R*、*G*、*B* 变换成 *Y*、C_1、C_2 的程序显示在 List 8.2 中。在 List 8.2 中，从亮度、色差信号变换到 *R*、*G*、*B* 信号时，为了不超过在内存中所显示的范围，需要进行数据范围的限定。



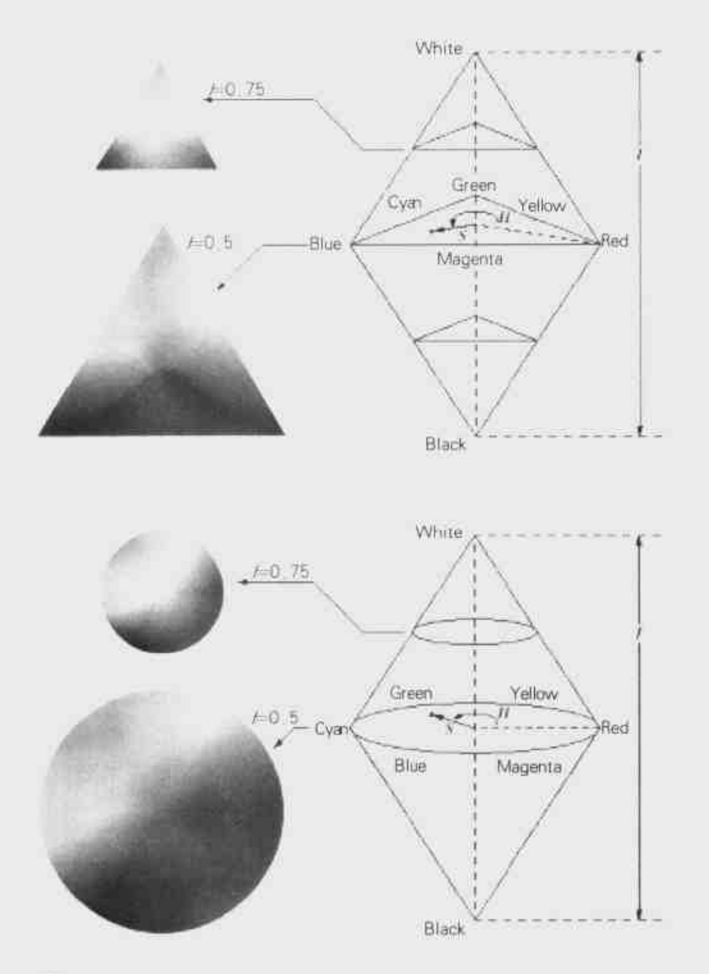


图 8-5

上述的色差信号与色调、饱和度之间的关系如图 8–6 所示，与图 8–5 所示的垂直于亮度轴线方向上的投影平面（即彩色圆）是一致的。从图 8–6 中可看出，色调 *H* 表示以色差信号 *B*–*Y*（即 C_1）为基准的坐标轴开始旋转了多少角度，饱和度 *S* 表示离开原点多大的距离。色调、饱和度与色差的关系可以用公式表示如下：

$$H = \tan^{-1}(C_1/C_2) \qquad (8.3)$$

$$S = \sqrt{C_1^2 + C_2^2}$$

相反，从色调 *H*、饱和度 *S* 变换到色差信号的公式如下：



$$C_1 = S \times \sin H \quad (8.4)$$
$$C_2 = S \times \cos H$$

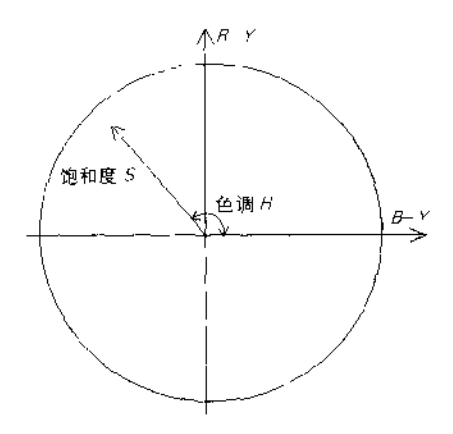


图 8-6

这些变换的程序显示在 List 8 3 中。色调 H 在 0° ~360° 的范围内，无色差信号时 H 的值不确定，这时可以将其代入 0 值。主要颜色的色调 H 和饱和度 S 显示在表 8–1 和表 8–2 中。表 8–1 表示白色被定义为 255 时的情况，也就是一般图像数据的表示形式。表 8–2 表示白色被定义为 1 时的情况。

从表 8–1 可以看出，颜色变化时色调值也变化，颜色不同则饱和度也不同。

表 8-1 彩条的色调 H 和饱和度 S（亮度 Y: 0~255）

	R (255,0, 0)	Ye (255, 255, 0)	G (0, 255, 0)	Cy (0,255, 255)	B (0, 0, 255)	Mg (255,0,255)
H	113 2°	173.0°	225.0°	293 2°	353 0°	45 0°
S	194 20	228 68	212 77	194 20	228 68	212 77

表 8-2 彩条的色调 H 和饱和度 S（亮度 I: 0~1）

	R (1,0,0)	Ye (1,1,0)	G (0,1,0)	Cy (0, 1, 1)	B (0,0,1)	Mg (1,0,1)
H	113 2°	173 0°	225.0°	293 2°	353.0°	45.0°
S	0.76	0 90	0 83	0.76	0 90	0.83

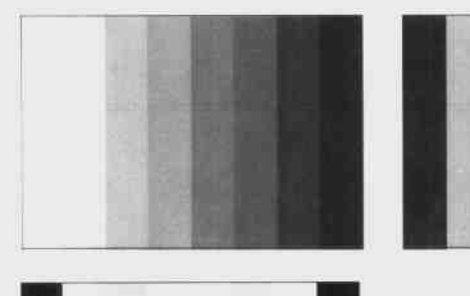
把彩色图像的 R、G、B 信息变换为亮度、色调、饱和度信息，将亮度信号图像化得到的就是灰度图像。色调和饱和度是把这些信号的差作为灰度差进行图像化，程序表示在 List 8.4 中。

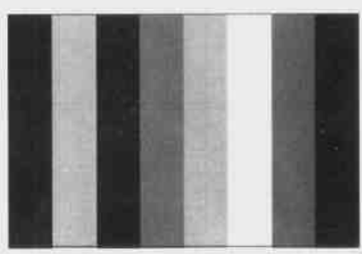


色调的表示是从某基准颜色开始计算（用在 0°～180° 之间的旋转角度来表示），当与基准颜色相同（色调的旋转角为 0°）时为 255，相对方向的补色（色调的旋转角为 180°）时为 0，0°～180° 之间用 254 级的灰度表示。在色调的表示中，当饱和度为 0（即无颜色信号）时将不计算色调，常常给予 0 灰度级。饱和度的图像是将饱和度的最小值作为像素的最小值 0，将饱和度的最大值作为像素的最大值 255，依次按比例将饱和度的数据转换为图像数据。

使用这些程序，可以从先前制作的彩条图像中取出亮度、色调、饱和度等各个信号。表示亮度信号的图像被显示在图 8-7（a）中，从左到右亮度是呈阶梯状变化的。图 8-7（b）是以 *R* 作为基准颜色表示的色调，R 的灰度级最高，接下来是 *Mg*，再接着是 *Ye*，*R* 的补色 *Cy* 的灰度级最低。图 8-7（c）表示彩条的饱和度，由于 *R*、*G*、*B* 的灰度被限定在 0～255 之间，饱和度的最大值因颜色的不同而不同。

(a) (b)
(c)





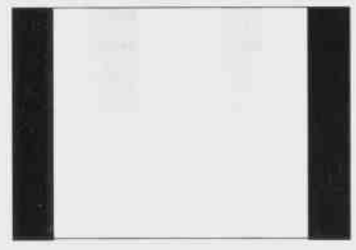


图 8-7

对实际摄像机拍摄的图像进行上述变换的结果如图 8-8 所示，其中，（a）是原始图像，（b）是亮度信号，（c）是以红色为基准的色调信号，（d）是饱和度信号。原始图像上，明亮的部分亮度信号值也高。原始图像的红色成分较多，由于色调信号以红色为基准，因此所表示的图像灰度级较高，图像整体偏亮。整个图像的颜色不是很深，所以饱和度信号偏暗，特别是人物头发的饱和度最低。

在实际应用中，可以有效地利用这些处理，比如设计服装时，可以参考表现色调和饱和度的图像，进行颜色调整，以便设计出不同风格的服装。



图 8-8

8.4 亮度、色调、饱和度的更改

正如前边所介绍的，彩色图像不是将其分解成 *R*、*G*、*B* 信号，而是分解成亮度信号（brightness）和色差信号（chrominance）来处理，这样才能更好地理解其颜色特征。所以，首先把彩色图像分解成亮度、色调、饱和度进行处理，然后再变换成 *R*、*G*、*B* 成分写入内存，就可以自由地改变彩色图像的颜色特征了。更改亮度、色调、饱和度的程序被显示在 List 8.5 中。在这个程序中输入的变量是通过 List 8.3 计算得到的亮度信号、色调、饱和度等数据，以及各个变更量。更改亮度、色调、饱和度后，需要计算出相应的色差信号 *C1* 和 *C2*，再由色差信号计算出 *R*、*G*、*B* 三基色（three primary color）进行表示。色差信号和三基色的计算程序分别显示在 List 8.6 和 List 8.7 中。



使用这个程序处理一个简单的例子，将图像的色调旋转一下，然后再从亮度、色差信号返回到 *R*、*G*、*B* 信号，那么就能够改变彩色图像的颜色，如图 8-9 所示。其中，（a）是原始图像，（b）是色调增加 120° 后的图像。由于在这个例子中旋转了整个图像的色调（旋转角为 120°），整体的颜色发生了变化，如图中人物服装的颜色由红色变为绿色，绿色变成了蓝色，蓝色变成了红色，粉红色变成了浅绿色，浅蓝色变成了粉红色等。如果把改变颜色的部分取出处理，则能改变所穿衣服和裙子的颜色。

(a) | (b)





图 8-9

List 8.1　彩条制作

```
#include "StdAfx.h"
#include "BaseList.h"
/*---- Colorbar ---彩条制作 ------------------------------------------
   image_r:    输出图像数据 R 分量指针
   image_g:    输出图像数据 G 分量指针
   image_b:    输出图像数据 B 分量指针
   xsize:      图像宽度
   ysize:      图像高度
   level:      亮度值
------------------------------------------------------------------*/
void Colorbar(BYTE *image_r, BYTE *image_g, BYTE *image_b,
int xsize, int ysize, int level)
{
   int     i, j, width;

   width = xsize / 8;
```



```
    for (j = 0; j < ysize; j++){
      for (i = 0; i < xsize; i++){
        if (((i >= 0) && (i < 2*width)) ||  //R
            ((i >= 4*width) && (i < 6*width)))
          *(image_r + j*xsize + i) = level;
        else *(image_r + j*xsize + i) = 0;
        if ((i >= 0) && (i < 4*width ))    //G
          *(image_g + j*xsize + i) = level;
        else *(image_g + j*xsize + i) = 0;
        if (((i >= 0) && (i < width )) ||  //B
            ((i >= 2*width) && (i < 3*width)) ||
            ((i >= 4*width) && (i < 5*width)) ||
            ((i >= 6*width) && (i < 7*width)))
          *(image_b + j*xsize + i) = level;
        else *(image_b + j*xsize + i) = 0;
      }
    }
  }
```

List 8.2　R、G、B与Y、R-Y、B-Y之间的变换

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Rgb_to_yc --- 由R、G B变换为亮度、色差信号 ----------------------
   image_r:     输入图像数据R分量指针
   image_g:     输入图像数据G分量指针
   image_b:     输入图像数据B分量指针
   y:           输出数据指针 Y
   c1:          输出数据指针 R-Y
   c2:          输出数据指针 B-Y
   xsize:       图像宽度
   ysize:       图像高度
------------------------------------------------------------------*/
void Rgb_to_yc(BYTE *image_r, BYTE *image_g, BYTE *image_b,
 int *y, int *c1, int *c2, int xsize, int ysize)
{
  int   i, j;
  float fr, fg, fb;
```



```
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        fr = (float)(*(image_r + j*xsize + i));
        fg = (float)(*(image_g + j*xsize + i));
        fb = (float)(*(image_b + j*xsize + i));
        *(y + j*xsize + i) = (int)(0.3 * fr + 0.59 * fg + 0.11 * fb);
        *(c1 + j*xsize + i) = (int)(0.7 * fr - 0.59 * fg - 0.11 * fb);
        *(c2 + j*xsize + i) = (int)(-0.3 * fr - 0.59 * fg + 0.89 * fb);
      }
    }
}
```

List 8.3　R-Y、B-Y与色调H、饱和度S之间的变换

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
/*---- C_to_SH --- 由色差信号计算饱和度和色调 ------------------------------
    c1:       输入数据指针 R-Y
    c2:       输入数据指针 B-Y
    sat:      饱和度数据指针
    hue:      色调数据指针
    xsize:    数列宽度
    ysize:    数列高度
--------------------------------------------------------------------------*/
void C_to_SH(int *c1, int *c2, int *sat, int *hue, int xsize, int ysize)
{
    int   i, j;
    float fhue, length;

    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        length=(float)(*(c1+j*xsize + i))*(float)(*(c1 + j*xsize+i))
            +(float)(*(c2+j*xsize+i))*(float)(*(c2 +j*xsize + i));
        *(sat + j*xsize + i) = (int)(sqrt((double)length));
        if (*(sat + j*xsize + i) > THRESHOLD){
          fhue = (float)(atan2((double)(*(c1 + j*xsize + i)),
```



```
            (double)(*(c2 - j*xsize + i))) * 180.0 / PI);
          if (fhue < 0 ) fhue = fhue + (float)360.0;
          *(hue + j*xsize + i) = (int)fhue;
          }
        else *(hue + j*xsize + i) = (int)NONE; //彩度小于阈值
      }
    }
  }
```

List 8.4　色调、饱和度的图像化

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
/*--- Hue_to_image ---由色调数据变换灰度图像-----------------------------
   sat:         饱和度数据指针
   hue:         色调数据指针
   stdhue:      基准色相值
   image_out:   输出图像数据
   xsize:       数列宽度
   ysize:       数列高度
---------------------------------------------------------------*/
void Hue_to_image(int *sat, int *hue, double stdhue,
BYTE *image_out, int xsize, int ysize)
{
   int    i, j;
   int    ihue;
   double  delt;

   for (j = 0; j < ysize; j++){
     for (i = 0; i < xsize; i++){
       if (*(sat - j*xsize + i) > 0){
         delt = fabs((double)(*(hue + j*xsize + i)) - (double)stdhue);
         if (delt > 180.0) delt = 360.0 - delt;
         ihue = (int)(255.0 - delt * 255.0 / 180.0);
         *(image_out + j*xsize + i) = (BYTE)ihue;
       }
       else *(image_out + j*xsize + i)=0;
```



```
      }
    }
}
/*--- Sat_to_image---由饱和度数据变换灰度图像 --------------------------
    sat:          饱和度数据指针
    image_out:    输出图像数据指针
    xsize:        数列宽度
    ysize:        数列高度
--------------------------------------------------------------------*/
int Sat_to_image(int *sat, BYTE *image_out, int xsize, int ysize)
{
    int i, j;
    int min, max;
    int isat;

    min = 255;
    max = 0;
    for (j = 0; j < ysize; j++){
      for (i = 0; i < xsize; i++ ){
        if (*(sat + j*xsize + i) > max) max = *(sat + j*xsize + i);
        if (*(sat + j*xsize + i) < min) min = *(sat + j*xsize + i);
      }
    }
    if (min == max) return -1;
    for (j = 0; j < ysize; j++){
      for (i = 0; i < xsize; i++ ){
        isat = 255 * (*(sat + j*xsize + i) - min) / (max - min);
        *(image_out + j*xsize + i) = (BYTE)(isat);
      }
    }
    return 0;
}
```



List 8.5 亮度、色调、饱和度的调整

```
#include "StdAfx.h"
#include "BaseList.h"
/*--- Change_YSH---亮度、饱和度、色调的调整--------------------------------
   in_y:      输入数据指针（亮度）
   in_sat:    输入数据指针（饱和度）
   in_hue:    输入数据指针（色调）
   out_y:     输出数据指针（亮度）
   out_sat:   输出数据指针（饱和度）
   out_hue:   输出数据指针（色调）
   ym:        亮度乘数
   sm:        彩度乘数
   hd:        色相乘数
   xsize:     数列宽度
   ysize:     数列高度
----------  -------------------------------------------------------------*/
void Change_YSH(int *in_y, int *in_sat, int *in_hue, int *out_y,
 int*out_sat,int*out_hue,float ym,float sm,float hd,int xsize, int ysize)
{
   int i, j;

   for (j = 0; j < ysize; j++){
     for (i = 0; i < xsize; i++){
       *(out_y + j*xsize + i)   = (int)(*(in_y + j*xsize + i) * ym);
       *(out_sat + j*xsize + i) = (int)(*(in_sat + j*xsize + i) * sm);
       *(out_hue + j*xsize + i) = (int)(*(in_hue + j*xsize + i) + hd);
       if(*(out_hue + j*xsize + i)>360)
         *(out_hue + j*xsize + i) = *(out_hue + j*xsize + i) - 360;
       if(*(out_hue + j*xsize + i)< 0)
         *(out_hue + j*xsize + i) = *(out_hue + j*xsize + i) + 360;
     }
   }
}
```

List 8.6 由饱和度和色调计算色差信号

```
#include "StdAfx.h"
```



```
#include "BaseList.h"
#include <math.h>

/*--- SH_to_C --- 由饱和度和色调计算色差信号 -----------------------------
   c1:        输出数据指针 R-Y
   c2:        输出数据指针 B-Y
   sat:       输入饱和度数据指针
   hue:       输入色调数据指针
   xsize:     数列宽度
   ysize:     数列高度
------------------------------------------------------------------*/
void SH_to_C(int *sat, int *hue, int *c1, int *c2, int xsize, int ysize)
{
   int   i, j;
   float rad;

   for (j = 0; j < ysize; j++) {
     for (i = 0; i < xsize; i++) {
       rad = (float)(PI * (*(hue + j*xsize + i)) / 180.0);
       *(c1+j*xsize+i)=(int)(*(sat + j*xsize + i) * sin((double)rad));
       *(c2+j*xsize+i)=(int)(*(sat + j*xsize + i) * cos((double)rad));
     }
   }
}
```

List 8.7　由亮度、色差变换*R*、*G*、*B*

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Yc_to_rgb --- 由亮度、色差变换 R、G、B 信号 ------------------
   y:         输入数据指针 Y
   c1:        输入数据指针 R-Y
   c2:        输入数据指针 B-Y
   image_r:   输出数据指针 R
   image_g:   输出数据指针 G
   image_b:   输出数据指针 B
```



```
  xsize:    图像宽度
  ysize:    图像高度
--------------------------------------------------------------------------*/
void Yc_to_rgb(int *y, int *c1, int *c2, BYTE *image_r, BYTE *image_g,
BYTE *image_b, int xsize, int ysize)
{
  int i, j;
  int ir, ig, ib;

  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      ir = *(y + j*xsize + i) + *(c1 + j*xsize + i);
      if (ir > 255) ir = 255;
      if (ir < 0) ir = 0;
      ig = (int)(*(y + j*xsize + i) - 0.3 / 0.59 *
        *(c1 + j*xsize + i) - 0.11 / 0.59 * (*(c2 + j*xsize + i)));
      if (ig > 255) ig = 255;
      if (ig < 0) ig = 0;
      ib = *(y + j*xsize + i) + *(c2 + j*xsize + i);
      if (ib > 255) ib = 255;
      if (ib < 0) ib = 0;
      *(image_r + j*xsize + i) = (BYTE)ir;
      *(image_g + j*xsize + i) = (BYTE)ig;
      *(image_b + j*xsize + i) = (BYTE)ib;
    }
  }
}
```



第 9 章

彩色分割

9.1 彩色信息

如第 8 章所述，彩色信息可分为亮度（brightness）、色调（hue）和饱和度（saturation）三个基本属性。在第 3 章中只讲到利用亮度（即灰度）信息来分割图像，可是有些情况只依靠亮度的差来提取物体区域是困难的，这是因为在同一物体上，由于光的照射程度不同也会产生明亮部分和阴暗部分。亮度因照射方式的不同而变化，而色调、饱和度则由于物体的不同而有特定的数值。虽然是灰度图像，但在图像上显示的是什么东西，大体上也能看得出来，但是如果在图像上加上颜色，那么可以看得更清楚一些，如红花、绿叶、蓝天等。颜色不仅使我们赏心悦目，而且为我们识别物体提供了重要的信息。在图像处理中不光要使用灰度信息，还要使用彩色信息，这样才能更精确地对图像进行分割。

9.2 颜色分布

对于灰度图像，是用灰度值进行阈值处理（thresholding）的，但是为了分割彩色图像中的特定颜色的区域，需要对彩色图像的 R、G、B 各个平面分别进行阈值处理。问题在于进行阈值处理时应该使用什么样的值作为阈值。最简单的方法是不断地改变阈值，查看处理的结果，然后再进行调整，不过如果知道了要提取区域 R、G、B 值的范围，还有更加有效的确定阈值的方法。首先来查看一下直方图，观察各个颜色的像素是如何分布的。

图 9–1 显示出了四种食品的彩色图像。图像在黑色的背景下显示出了橘红色的胡萝卜、绿色的黄瓜、黄色的橘子、白色的鸡蛋。考虑如何只把橘子分割出来。在图 9–2 的（a）～（c）中显示了彩色图像的 R、G、B 各个分量的直方图，其中，（a）是 R 平面，（b）是 G 平面，（c）是 B 平面。从这些直方图中可以发现，在 0～50 之间存在一个最大的峰，这个就是背景。然而，想要分割的橘子区域到底在哪个部分，从这些直方图中是不容易看出来的。虽然我们仅从 R、G、B 中某一个分量的直方图就可以知道画面中占面积比较大的部分的特征，但是占面积比较小的区域的分布“山峰”，经常会被掩盖在背景等大区域的“山峰”中，以致很难发现其分布的特征。

有没有更容易理解的像素的分布方式呢？答案是肯定的。如图 9–3 所示，R 与 G、R 与 B 等用两个分量来表示的分布图就是其中的一种。图 9–3 的（a）图和（b）图的横坐标都是 R 值，纵坐标分别是 G 值和 B 值。由于每个像素的颜色都表现为（R,G,B）一组值，所以每个像素的颜色在图 9–3 的（a）图和（b）图上各对应一个点，其中，（a）是 R 与 G 的关系，（b）是 R 与 B 的关系。在图 9–2 所示的直方图中，用柱形的长度来表示出现的频数，而图 9–3 的直方



图中，用明暗的点表示出现的频数，越明亮的地方表示出现的频数越高。图 9–3 的直方图是在二维平面上表现频数分布的，因此被称为二维直方图（two–dimensional histogram）。当使用这个二维直方图时，就可以清晰地观察出颜色空间上的像素分布。在 R、G、B 的各个直方图中不能区分的区域，通过二维直方图，则能够清楚地区分开。计算二维直方图并使其图像化的程序显示在 List 9.1 中。

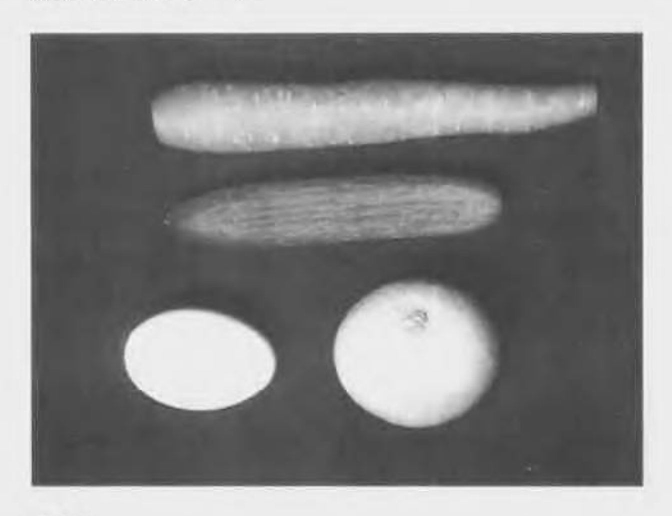
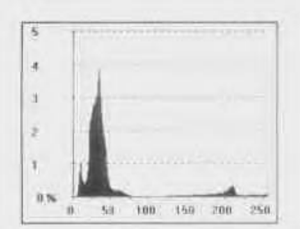
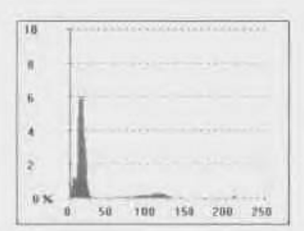


图 9-1

(a)|(b)|(c)





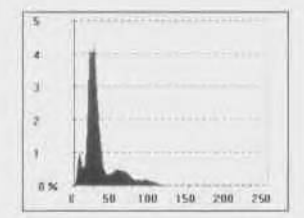


图 9-2

(a) | (b)



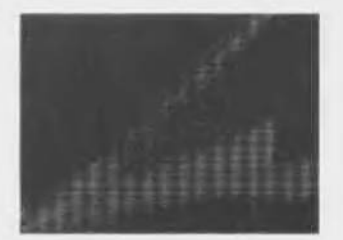


图 9-3



9.3 基于颜色分布提取物体

通过上面的方法试着把图 9-1 中的橘子提取出来。在这个图像中除了橘子以外，还有黄瓜、胡萝卜、鸡蛋。若是光凭明暗来区分这些食品是非常困难的。在此，可以利用绿色、橘红色、黄色、白色等颜色的差异来分割图像，以达到提取橘子的目的。

首先注意 R 值，把具有 R≥125 范围内的像素提取出来。其结果如图 9-4（a）所示，用 *R* 值进行的阈值处理把背景和黄瓜除去了，留下了三种食品——胡萝卜、橘子和鸡蛋。接着用 *G* 值进行阈值处理，对 9.4（a）中的图像提取 G≥125 范围内的像素，如图 9-4（b）所示，胡萝卜的大部分像素都被除去了。进一步用 *B* 值进行阈值处理，提取 B≤90 范围内的像素，如图 9-4（c）所示，橘子的绝大部分被提取出来，只是橘子的顶部被除去，因为其颜色不是黄色。

由此可见，仅用单一颜色信息还是不能分割复杂的彩色图像的，使用三个颜色信息就能够提取出所要求的目标区域。给出 R、G、B 的最大值和最小值，通过阈值处理提取在该范围内的像素。该程序被显示在 List 9.2 中。

(a)|(b)|(c)

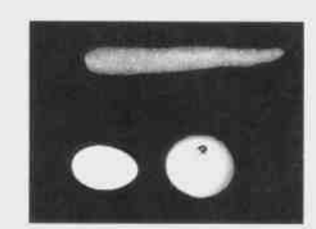
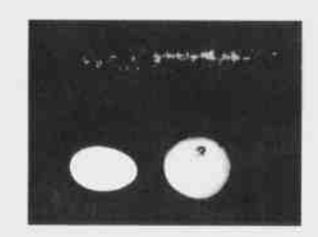
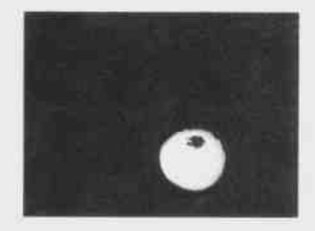




图 9-4

上面所举的例子是根据 R、G、B 三基色的大小，对原图像进行区域分割与提取的，但是在许多情况下，仅根据 R、G、B 的大小很难对物体进行提取，因为 R、G、B 的相关性太强，也就是说在改变 R、G、B 某个分量的大小时，其他分量也会跟着变化。所以，有时候利用第 8 章所介绍的，先将 R、G、B 变换成相关性较低的 H、S、I，利用 H、S、I 进行区域提取的效果可能会更好些，不过需要花费更多的处理时间。

9.4 图像合成

在制作电视节目时，所使用的合成法中有被称为 chroma key 的方法。chroma key 的含义是：基于颜色（chroma）的差异做成用于合成的键（key）信号。chroma key 的原理如图 9-5 所示。在 chroma key 中，从拍摄的蓝色背景前站立着人物的图像中，分离蓝色的背景部分，做成用于

合成的键信号。对于这个键信号，比如前景 255、背景 0 那样分配各个像素的值。依据这个键信号，通过把图像 A 的蓝色背景部分用其他的图像 B 替代，这样就完成了图像合成（image mixing）。

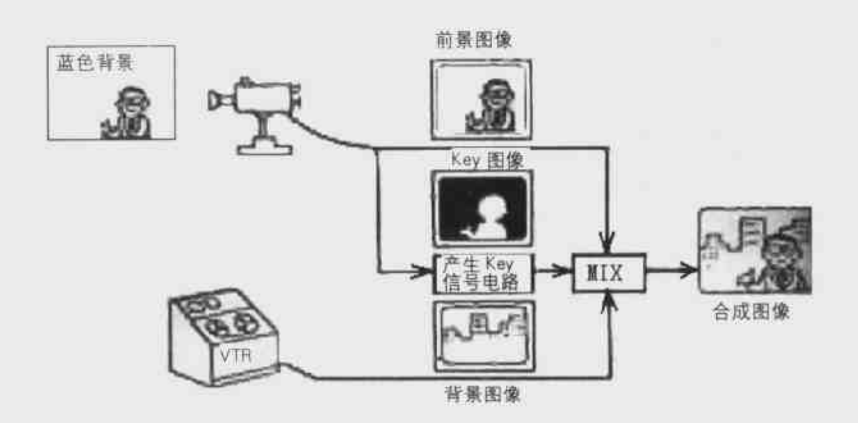


图 9-5

在实际的 chroma key 中，不仅可以使用蓝色，还可以用其他的颜色作为背景来提取，自然图像合成有各种各样的技巧。在此，可以采用简单的图像处理来试一试类似 chroma key 的效果。首先对于各个像素计算下式的值：

$$(R+G)/2-B \tag{9.1}$$

用这个值的大小来提取背景。蓝色时这个值为负数，也就意味着像素的这个值越小就越接近蓝色。合成用的键图像是通过对每个像素用式（9.1）进行计算并对计算的结果进行阈值处理后获得的。由以上方法生成键图像的程序被显示在 List 9.3 中。在这个程序中前景部分设为 255（白）、背景部分为 0（黑）。接着基于这个键图像，把值为 255 的部分用前景图像的像素代替，值为 0 的部分用其他图像的像素值替代，从而将两幅图像合成为一幅图像。这个合成程序显示在 List 9.4 中。

可以用这个程序进行实际图像的合成。如图 9-6 所示，拍摄的是一幅在浅蓝色的 chroma key 背景前面一位女播音员的图像，图 9-7 拍摄的是动物园中两只熊猫的图像，要求把图 9-6 中的女播音员作为前景，图 9-7 的两只熊猫的图像作为背景来合成两幅图像。那么，首先使用 List 9.3 的程序除去图 9-6 中浅蓝色背景得到键图像，如图 9-8 所示，阈值是-18。接着采用 List 9.4 的程序对前景图像（图 9-6）和背景图像（图 9-7）进行合成，合成时，键图像的白色部分镶嵌前景图像，黑色部分镶嵌背景图像，合成结果如图 9-9 所示。在合成后的图像中，原来女播音员浅蓝色的背景被置换成了大熊猫的图像，就好像女播音员站在熊猫馆里津津乐道



地介绍着大熊猫的生活一样。



图 9-6



图 9-7

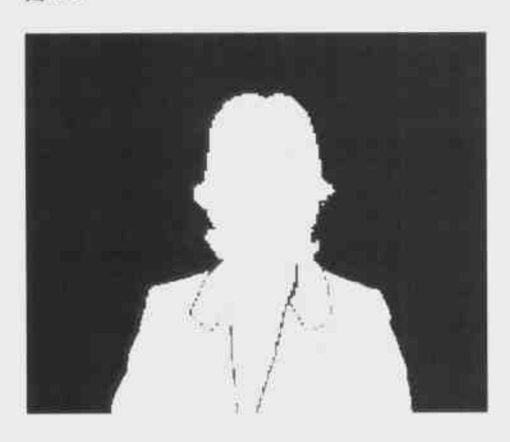


图 9-8

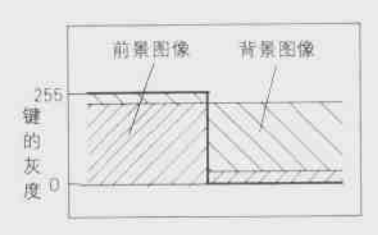


图 9-9

至此，图像的合成就完成了。但是，得到的合成图像，还是让人觉得有一点人工合成、不太自然的感觉。这是因为在女播音员头发的轮廓线以及衣服领口等处的边界有锯齿状或蓝色框存在。如果想使不确切形状的头发和领口自然地合成，就需要提取浅蓝色的灰度变化，这时就不能采用如图 9–10（a）那样的二值键信号了，而需要采用如图 9–10（b）所示的多值键信号。多值键信号的交叉部分不同于图 9–10（a）所示的样子，或者前景、背景的二值，而是具有 0~255 多层次的灰度级。合成时依据这个灰度级，在图像上的不同位置改变混合比。如图 9–10（b）所示，具有多值灰度级的键信号被称为软键信号。而图 9–10（a）所示的二值灰度级的键则被称为硬键信号。



(a) | (b)



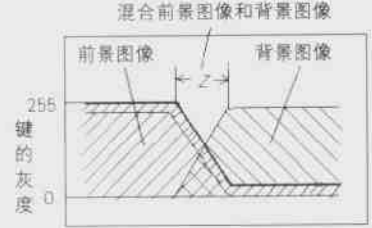


图 9-10

生成软键信号的程序显示在 List 9.5 中。图 9–11 所示的图像是使用这个程序从图 9–6 的图像生成的软键信号，阈值是最小值–100，最大值 0。

与硬键信号相比，使用这个软键信号得到了相对自然的图像合成，但合成面还可以见到微微蓝色，仍然有不自然的部分。为此，可以事先消除相当于前景图像的边界处 *Z* 部分（图 9–10（b）所示）的蓝色，然后再进行合成，这样不仅能够防止边界处出现蓝色框，而且人物的影子也可能在背景图像中很自然地合成。消除蓝色的简单方法是：在边界处使用与亮度相近的 G 值来替代 B 值。根据这个方法，进行图像合成的程序被显示在 List 9.6 中。图 9–12 是用这个程序由软键信号进行合成的结果。与图 9–9 中基于硬键信号的图像合成结果相比，基于软键信号的图像合成中，女播音员头发的轮廓线、衣服领口等处的锯齿状或蓝色框消失了，前景图像和背景图像更加自然地融合成一体。在实际的电视节目中还会使用更加复杂的处理，在此只是简单介绍基本的思路。



图 9-11



图 9-12

chroma key 的背景之所以用蓝色，是因为从背景中分离的对象往往是人，蓝色和人皮肤的



颜色基本上是互补色（色调相反），所以容易把人从蓝色背景里分离出来。不过对于蓝色眼睛的人种，眼睛部位会被合成进背景图案，产生奇怪的效果。为了防止这种情况的发生，在国外，有些节目主持人会带上变色的隐形眼镜，来改变眼睛的颜色。在使用 chroma key 时，如果背景前物体的颜色呈现蓝色成分较多的紫色或者青绿色，提取前景时就会不稳定，这时一般会通过人物的服装以及小道具的颜色进行调整。

List 9.1　二维直方图

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Hist2 image --- 计算二维直方图并图像化 ----------------------
  image_in1:    输入 x 轴图像数据
  image_in2:    输入 y 轴图像数据
  image_hist:   输出二维直方图图像
  xsize:        输入图像宽度
  ysize:        输入图像高度
---------------------------------------------------------------*/
void Hist2_image(BYTE*image_in1,BYTE*image_in2,BYTE*image_hist,int xsize,int ysize)
{
  int   i, j;
  float kx, ky;
  int   hx, hy, max, kk;

  for ( j = 0; j < ysize; j++)          //初始化
    for (i = 0; i < xsize; i++)
      *(image_hist + j*xsize + i) = 0;
  max = 0;

  ky = (float)256 / (float)ysize;
  kx = (float)256 / (float)xsize;

  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      hy = (int)((float)(HIGH - *(image_in2 + j*xsize + i)) / ky);
      hx = (int)((float)(*(image_in1 + j*xsize + i)) / kx);
      if (*(image_hist + hy*xsize + hx) < HIGH)
```



```
        *(image_hist + hy*xsize + hx)=*(image_hist+hy*xsize+hx)+1;
      if(max<*(image_hist+hy*xsize+hx)) max=*(image_hist+hy*xsize+hx);
    }
  }
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      if (*(image_hist + j*xsize + i) != 0) {
        kk = (long)(*(image_hist + j*xsize + i)) * HIGH / max + BIAS;
        if (kk > HIGH)  *(image_hist + j*xsize + i) = HIGH;
        else      *(image_hist + j*xsize + i) = kk;
      }
    }
  }
  for(j=0; j<ysize; j++)*(image_hist+j*xsize+1) = HIGH;       //y 轴
  for(i=0; i<xsize;i++)*(image_hist+(ysize-2)*xsize+i)=HIGH;  //x 轴

}
```

List 9.2 基于RGB的阈值处理

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Thresh_color --- R,G,B 的阈值处理 ----------------------------------
   image_in_r:      输入图像 R 分量数据指针
   image_in_g:      输入图像 G 分量数据指针
   image_in_b:      输入图像 B 分量数据指针
   image_out_r:     输出图像 R 分量数据指针
   image_out_g:     输出图像 G 分量数据指针
   image_out_b:     输出图像 B 分量数据指针
   thdrl, thdrm:    R 阈值 (min,max)
   thdgl, thdgm:    G 阈值 (min,max)
   thdbl, thdbm:      B 阈值 (min,max)
   xsize:             图像宽度
   ysize:             图像高度
---------------------------------------------------------------*/
void Thresh_color(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
```



```
BYTE *image_out_r, BYTE *image_out_g, BYTE *image_out_b,
int thdrl, int thdrm, int thdgl, int thdgm, int thdbl, int thdbm,
int xsize, int ysize)
{
  int  i, j;

  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      *(image_out_r + j*xsize + i) = *(image_in_r + j*xsize + i);
      *(image_out_g + j*xsize + i) = *(image_in_g + j*xsize + i);
      *(image_out_b + j*xsize + i) = *(image_in_b + j*xsize + i);
    }
  }
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      if (*(image_out_r + j*xsize + i) < thdrl)
      {
        *(image_out_r + j*xsize + i) = 0;
        *(image_out_g + j*xsize + i) = 0;
        *(image_out_b + j*xsize + i) = 0;
      }
      if (*(image_out_r + j*xsize + i) > thdrm)
      {
        *(image_out_r + j*xsize + i) = 0;
        *(image_out_g + j*xsize + i) = 0;
        *(image_out_b + j*xsize + i) = 0;
      }
      if (*(image_out_g + j*xsize + i) < thdgl)
      {
        *(image_out_r + j*xsize + i) = 0;
        *(image_out_g + j*xsize + i) = 0;
        *(image_out_b + j*xsize + i) = 0;
      }
      if (*(image_out_g + j*xsize + i) > thdgm)
      {
        *(image_out_r + j*xsize + i) = 0;
        *(image_out_g + j*xsize + i) = 0;
```



```
          *(image_out_b + j*xsize + i) = 0;
        }
        if (*(image_out_b + j*xsize + i) < thdbl)
        {
          *(image_out_r + j*xsize + i) = 0;
          *(image_out_g + j*xsize + i) = 0;
          *(image_out_b + j*xsize + i) = 0;
        }
        if (*(image_out_b + j*xsize + i) > thdbm)
        {
          *(image_out_r + j*xsize + i) = 0;
          *(image_out_g + j*xsize + i) = 0;
          *(image_out_b + j*xsize + i) = 0;
        }
      }
    }
}
```

List 9.3 硬键信号的生成

```
#include "StdAfx.h"
#include "BaseList.h"

/* - hard_key --- 生成合成键 -------------------------------
   image_in_r:   输入图像 R 分量数据指针
   image_in_g:   输入图像 G 分量数据指针
   image_in_b:   输入图像 B 分量数据指针
   image_key:    输出合成键数据指针
   thresh:       阈值
---------------------------------------------------------*/
void Hard_key(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
 BYTE *image_key, int xsize, int ysize, int thresh)
{
   int  i, j, d;

   for (j = 0; j < xsize; j++) {
     for (i = 0; i < ysize; i++) {
```



```
        d=((int)(*(image_in_r+j*ysize+i))+(int)(*(image_in_g+j*ysize+ i)))/2
          - (int)(*(image_in_b + j*ysize + i));
        if (d >= thresh) *(image_key + j*ysize + i) = 255;
        else            *(image_key + j*ysize + i) = 0;
      }
    }
}
```

List 9.4 图像合成

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- synth --- 图像合成 ----------------------------------------
    image_in1_r:    输入前景图像 R 分量数据指针
    image_in1_g:    输入前景图像 G 分量数据指针
    image_in1_b:    输入前景图像 B 分量数据指针
    image_in2_r:    输入背景图像 R 分量数据指针
    image_in2_g:    输入背景图像 G 分量数据指针
    image_in2_b:    输入背景图像 B 分量数据指针
    image_out_r:    输出合成图像 R 分量数据指针
    image_out_g:    输出合成图像 G 分量数据指针
    image_out_b:    输出合成图像 B 分量数据指针
    image_key:      输入合成键图像数据指针
-----------------------------------------------------------------*/
void Synth(BYTE *image_in1_r, BYTE *image_in1_g,
        BYTE *image_in1_b, BYTE *image_in2_r,
        BYTE *image_in2_g, BYTE *image_in2_b,
        BYTE *image_out_r, BYTE *image_out_g,
        BYTE *image_out_b, BYTE *image_key,
      int xsize, int ysize)
{
    int   i, j;
    int   rr1, gg1, bb1;
    int   rr2, gg2, bb2;
    long  kk;
```



```
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        rr1 = (int)(*(image_in1_r + j*xsize + i));
        gg1 = (int)(*(image_in1_g + j*xsize + i));
        bb1 = (int)(*(image_in1_b + j*xsize + i));
        rr2 = (int)(*(image_in2_r + j*xsize + i));
        gg2 = (int)(*(image_in2_g + j*xsize + i));
        bb2 = (int)(*(image_in2_b + j*xsize + i));
        kk = (long)(*(image_key + j*xsize + i));
        *(image_out_r + j*xsize+i)=(unsigned char)((rr1*kk+rr2*(255-kk))/255);
        *(image_out_g + j*xsize + i) = (unsigned char)((gg1*kk+gg2*(255-kk))/255);
        *(image_out_b + j*xsize + i) = (unsigned char)((bb1*kk+bb2*(255-kk))/255);
      }
    }
}
```

List 9.5　用于合成的软键信号生成

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Soft_key --- 生成软合成键 ---------------------------
    image_in_r:     输入图像 R 分量数据指针
    image_in_g:     输入图像 G 分量数据指针
    image_in_b:     输入图像 B 分量数据指针
    image_key:      输出合成键图像数据指针
    thdh, thdl:     阈值(max,min)
-----------------------------------------------------------*/
void Soft_key(BYTE *image_in_r, BYTE *image_in_g, BYTE *image_in_b,
 BYTE *image_key, int xsize, int ysize, int thdh, int thdl)
{
    int i, j, d;
    int kk;

    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
            d=((int)(*(image_in_r+j*xsize+i)) + (int)(*(image_in_g + j*xsize + i)))/ 2
```



```
                - (int)(*(image in b + j*xsize + i));
            kk = ((long)(d - thdl) * 255 / (thdh - thdl));
            if (kk > 255)      *(image_key + j*xsize + i) = 255;
            else if (kk < 0) *(image_key + j*xsize + i) = 0;
        else                   *(image_key + j*xsize + i) = kk;
      }
    }
}
```

List 9.6 图像合成（消除边界处颜色）

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- S_synth --- 图像合成（消除境界线）---------------------
   image_in1_r:     输入前景图像 R 分量数据指针
   image_in1_g:     输入前景图像 G 分量数据指针
   image_in1_b:     输入前景图像 B 分量数据指针
   image_in2_r:     输入背景图像 R 分量数据指针
   image_in2_g:     输入背景图像 G 分量数据指针
   image_in2_b:     输入背景图像 B 分量数据指针
   image_out_r:     输出合成图像 R 分量数据指针
   image_out_g:     输出合成图像 G 分量数据指针
   image_out_b:     输出合成图像 B 分量数据指针
   image_key:       合成键图像分量数据指针
-----------------------------------------------------------*/
void S_synth(BYTE *image_in1_r, BYTE *image_in1_g, BYTE *image_in1_b,
        BYTE *image_in2_r, BYTE *image_in2_g, BYTE *image_in2_b,
        BYTE *image_out_r, BYTE *image_out_g, BYTE *image_out_b,
     BYTE *image_key, int xsize, int ysize)
{
   int  i, j;
   int  rr1, gg1, bb1;
   int  rr2, gg2, bb2;
   long  kk;

   for (j = 0; j < ysize; j++) {
     for (i = 0; i < xsize; i++) {
```



```
            rr1 = (int)(*(image_in1_r + j*xsize + i));
            gg1 = (int)(*(image_in1_g + j*xsize + i));
            bb1 = (int)(*(image_in1_b + j*xsize + i));
            rr2 = (int)(*(image_in2_r + j*xsize + i));
            gg2 = (int)(*(image_in2_g + j*xsize + i));
            bb2 = (int)(*(image_in2_b + j*xsize + i));
            kk = (long)(*(image_key + j*xsize + i));
            if (kk == 255 || kk == 0) {       // 前景或背景
        *(image_out_r+j*xsize+i)= (BYTE)((rr1*kk+rr2*(255-kk))/255);
            *(image_out_g+ j*xsize + i) = (BYTE)((gg1*kk+gg2*(255-kk))/255);
            *(image_out_b+ j*xsize + i) = (BYTE)((bb1*kk+bb2*(255-kk))/255);
        }
        else {                                //境界部
            *(image_out_r+ j*xsize + i) = (BYTE)((gg1*kk+rr2*(255-kk))/255);
            *(image_out_g+ j*xsize + i) = (BYTE)((gg1*kk+gg2*(255-kk))/255);
            *(image_out_b+ j*xsize + i) = (BYTE)((gg1*kk+bb2*(255-kk))/255);
        }
      }
    }
}
```



- DataType　列的数据类型。
- Expression　用于创建该列的表达式，这个表达式可以计算列值，或者产生一个统计列（可参见 chapter7—2 示例）。
- MappingType　指定了该列在转换为 XML 时是如何被映射的，其取值如表 7—7 所示。

表 7-7　MappingType 属性的取值

取　值	说　明
Attribute	列被映射为 XML 属性
Element	列被映射为 XML 元素（默认值）
Hidden	列被映射为一个内部结构
SimpleContent	列被映射为 XML 文本

2. DataColumn 的属性

DataColumn 的属性如表 7—8 所示。

表 7-8　DataColumn 对象的属性

属　性	说　明
AllowDBNull	定义表中的列是否可为空，可以为空为 True，否则为 False。默认值为 True
AutoIncrement	定义列的值是否自动增加，为 True 时，自动增加，为 False 是不自动增加。默认值为 False。设为自动增加后，此列的值用户不能自己输入
AutoIncrementSeed	当 AutoIncrement 设为 True 时，设置列值的起始值
AutoIncrementStep	当 AutoIncrement 设为 True 时，设置每增一条新的记录后，列值的增量
Caption	设置列的标题，默认值为数据列 ColumnName 的属性值
ColumnName	设置或返回在 Columns 集合中列的名字
DataType	设置列的数据类型
DefaultValue	设置列的默认值
Expression	设置一个表达式，用于过滤行、计算列值或者生成一个统计列
ExtendedProperties	设置或返回用户的信息
Ordinal	返回该列在 Columns 集合中的位置
ReadOnly	该属性决定了在包含该列的行加入到表中之后，列的值是否可被修改。设为 True，不可修改，设为 False，可修改
Table	返回该列所属的表名
MaxLength	文本列的最大长度
Unique	决定表中的每一行是否必须有一个惟一的列值。是为 True，否为 False

从上表可以看出，列的属性的设置和建立数据库表时，列的属性设置含义是一样的。



10.1 关于几何变换

在电视上经常能够看到图 10-1 所示的图像，本章主要讲解如何使用几何变换来产生各种各样的图像效果。

如图 10-1 所示的变形在图像处理领域被称为几何变换（geometric transformation）。图 10-1 是通过透视变换（perspective transformation）处理的例子。使用几何变换的目的，是为了让电视节目看起来更富有渲染力，此外在其他场合，这方面的应用也很广泛。在天气预报中经常可以看到气象卫星所拍摄的云层图像，其实这也是经过几何变换所得到的图像。从人造卫星上用照相机拍摄的图像中包含有镜头引起的变形，需要通过几何变换进行校正，才能得到未变形的图像。

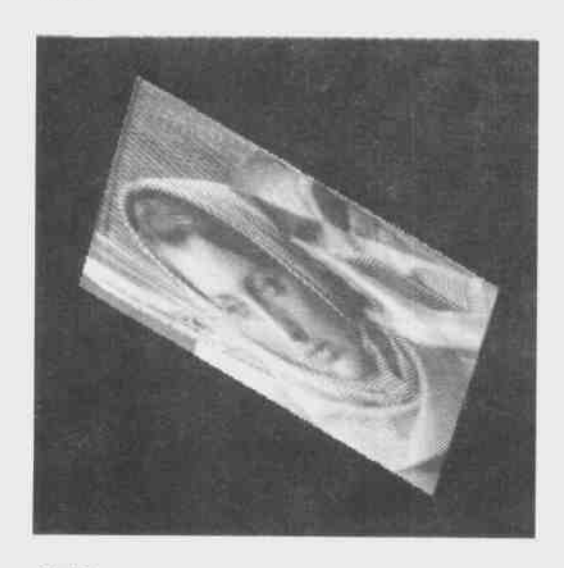


图 10-1

那么，几何变换是一种什么样的处理呢？旅游时为了尽可能把周围广阔的美景收录到镜头当中，有时需要用照相机拍摄数枚全景照片，然后把它们粘连起来，粘连是相当不容易的，会出现错位、弯曲，从而很难作到天衣无缝，为了使粘连达到最佳效果，必须采取放大或移位等操作。可见，几何变换是通过改变像素的位置来实现的。前面所讲到的方法都是改变灰度值的处理。几何变换中有放大缩小（dilation）、平移（translation）、旋转（rotation）等几种处理方式。下面以简单的例子进行说明。

首先，对本章中使用的坐标系统进行说明。通常图像处理的坐标系是使用第 2 章所述的，



以左上角为原点且向右及向下为正方向的坐标系统，但是使用这样的坐标系，在以原点为中心放大图像时，如图 10-2（a）所示，图像的范围只在右下方向外扩大。而以图像的正中间为中心放大图像时，其上下左右均等地向外扩大，感觉会更自然。因此，如图 10-2（b）所示，以图像的中心为原点的坐标系更方便，这就是本章所采用的坐标系。

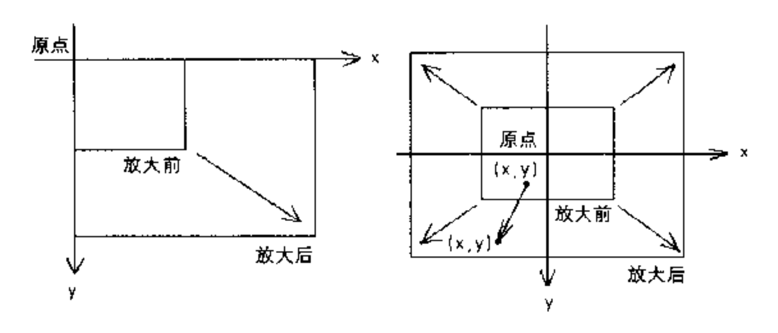


图 10-2

10.2 放大缩小

首先考虑改变图像的大小。如图 10-2（b）所示，某一点（x, y）经过放大缩小后其位置变为（X, Y），则两者之间有如下关系：

$$X = ax \qquad (10.1)$$
$$Y = by$$

a、b 分别是 x 方向、y 方向的放大率。a、b 比 1 大时放大，比 1 小时缩小。对于所有的像素点（x, y）进行计算，把输入图像上的点（x, y）的灰度值代入输出图像上的点（X, Y）处，就可以把图像放大或缩小了。

那么让我们实际操作一下吧。

记述这个处理的程序被显示在 List 10.1 中。在此，图像数据指针 image_in 及 image_out 的大小是宽 xsize 像素，高 ysize 像素，x 方向和 y 方向的范围分别是 $-xsize/2 \le x < xsize/2$ 和 $-ysize/2 \le y < ysize/2$。在以图像中心为坐标原点的坐标系中，x 和 y 是坐标，在以图像的左上角为座标原点的坐标系上需要加一个参数(xsize/2，ysize/2)，使坐标变为（x+xsize/2，y+ysize/2）。另外，为了防止由式（10.1）计算的结果 x 和 y 超出图像范围，产生程序暴走现象，在此进行了范围的检查。

使用这个程序缩小 1/2 的例子（$a=b=1/2$ 时）和放大 2 倍的例子（$a=b=2$ 时）分别显示在图 10-3（b）和（c）中。缩小 1/2 的图像似乎没有什么问题，但是放大 2 倍的图像有点不



太正常，到底是怎么回事呢？

(a) (b)
(c)





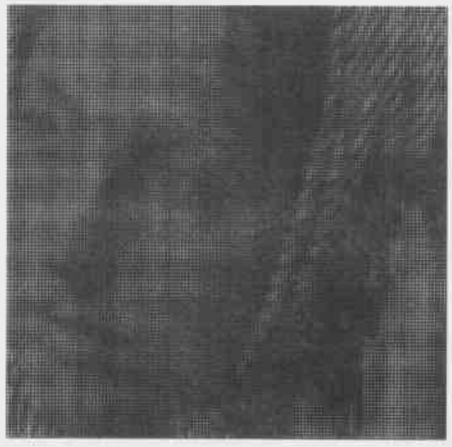
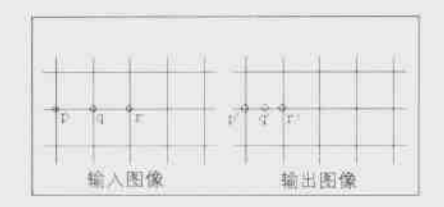


图 10-3

如图 10–4 所示，当输入图像的像素 p 对应于输出图像的像素 p'，输入图像上的 p 点的邻点 q，以及再下一个邻点 r 分别对应于输出图像上的 q'和 r'时，q'和 r'按照放大率或者接近 p'点，或者远离 p'点。缩小 1/2 时，如图 10–4（a）所示，q 点所对应 q'点的位置不在像素位置，这样在输出图像上将自动被取消，从而 p'和 r'点成为邻点。另一方面，放大 2 倍后，如图 10–4（b）所示，q'和 r'是相隔一个像素排列的，即输出图像上的 p'点的邻点，以及 q'点的邻点什么也没有写入。这就是图 10–3（c）中像素呈现断断续续状态的原因。



(a) | (b)



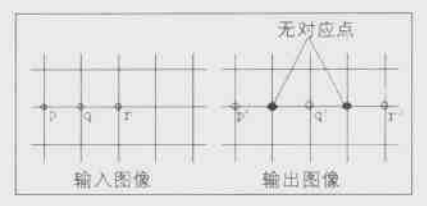


图 10-4

以上的做法是以输入图像为基准来查找输出图像上的对应点的，在放大时出现了输出图像上的一些位置没有对应像素值的情况。如果以输出图像为基准，对于输出图像上的每个像素，查找其在输入图像上的对应像素，就可以避免上述现象。为此，可以考虑式（10.1）的逆运算，即：

$$
\begin{aligned}
x &= X / a \\
y &= Y / b
\end{aligned}
\tag{10.2}
$$

如果对于输出图像上的所有像素（X，Y），用式（10.2）进行计算，求出对应输入图像上的像素（x，y），写入这个像素的灰度值，图 10–3（c）所示的现象就不会产生了。以这种方式进行放大缩小的程序显示在 List 10.2 中，所进行的缩小 1/2 和放大 2 倍的例子显示在图 10–5 中，其中，a——缩小 1/2，b——放大 2 倍，c——图像中心部分放大。在 List 10.2 中，对所计算的地址加上 0.5，以便对式（10.2）的计算结果进行四舍五入处理。

(a)|(b)|(c)





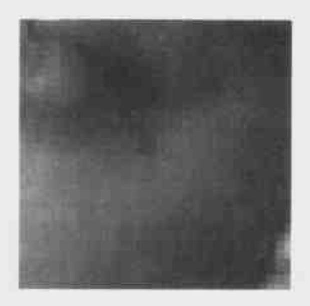


图 10-5

式（10.2）进行的是实数运算，x 和 y 包括小数位。然而，输入图像的像素地址必须是整数，所以对于地址计算，有必要采取某种形式对地址进行整数化。在此，经常用的整数化方式就是四舍五入的取整方法。如图 10–6 所示，在图像上就是选择最靠近坐标点（x，y）的方格上的点，从而被称为最近邻点法（nearest neighbor approach），也被称为零阶内插法（zero–order interpolation）。对于这种方法，从图 10–5（c）的放大图可以看出图像呈现马赛克状（mosaic），



这种现象放大率越大将越明显。

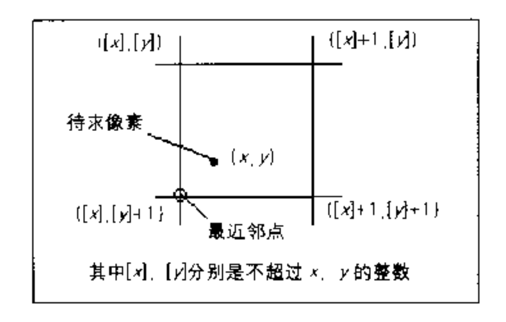


图 10-6

为了提高精度，可以采用被称为双线性内插（bilinear interpolation approach）的方法。这种方法是当所求的地址不在方格上时，求到相邻的 4 个方格上点的距离之比，用这个比率和 4 邻点（four nearest neighbors）像素的灰度值进行灰度内插，如图 10–7 所示。

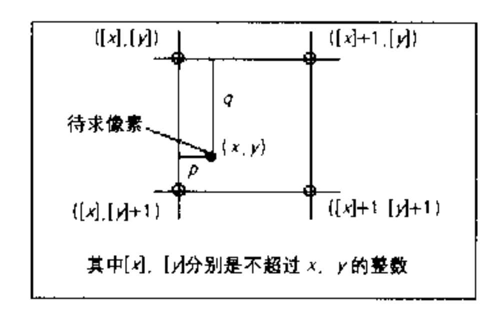


图 10-7

这个灰度值的计算式如下：

$$d(x,y) = (1-q)\{(1-q)\cdot d([x],[y]) + p\cdot d([x]+1,[y])\} + q\{(1-p)\cdot d([x],[y]+1) + p\cdot d([x]+1,[y]+1)\} \tag{10.3}$$

在此，$d(x, y)$ 表示坐标 (x, y) 处的灰度值，[x]和[y]分别是不超过 x 和 y 的整数值。实现这个方法的程序显示在 List 10.3 中。用双线性内插法处理的例子如图 10–8 所示，其中，(a) 是缩小 1/2，(b) 是放大 2 倍，(c) 是图像中心部分放大。10–8 (c) 的放大图也没有呈现马赛克状，而是显现很平滑。



(a) (b)
(c)







图 10-8

这种双线性内插法不仅可采用上述的 4 邻点，也可采用 8 邻点、16 邻点、24 邻点进行高次内插，在此不作具体说明。此外，为了精确地进行缩小，还需要进行一些前置滤波处理，在这里也不作更详细的解释。

10.3 平移

下面来分析图像位置的移动。如图 10-9 所示，为了使图像分别沿 x 轴和 y 轴向右、下平移 x_0 和 y_0，需要采用如下的平移（translation）变换公式：

$$X = x + x_0$$
$$Y = y + y_0 \tag{10.4}$$

逆变换公式如下所示：

$$x = X - x_0 \quad y = Y - y_0 \tag{10.5}$$

实现这个平移变换的程序显示在 List 10.4 中，处理示例如图 10–10 所示。在 List 10.4 中采用的是双线性内插法，所以能够以 1 像素以下的精度数值实现图像平移。

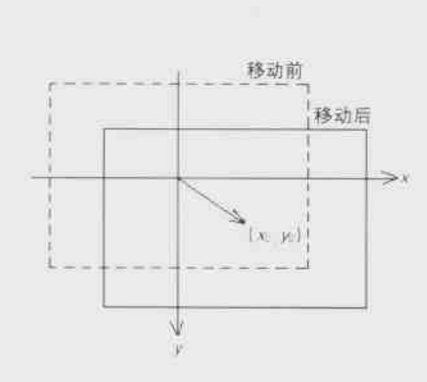


图 10-9



图 10-10

10.4 旋转

下面考虑一下旋转图像（rotation image），如图 10–11 所示使图像逆时针旋转 $\theta°$ 需要如下的变换公式

$$X = x\cos\theta + y\sin\theta \quad Y = -x\sin\theta + y\cos\theta \tag{10.6}$$

逆变换公式如下所示：

$$x = X\cos\theta - Y\sin\theta \quad y = X\sin\theta + Y\cos\theta \tag{10.7}$$

实现这个旋转变换（rotation transform）的程序显示在 List 10.5 中，处理示例如图 10–12 所示。

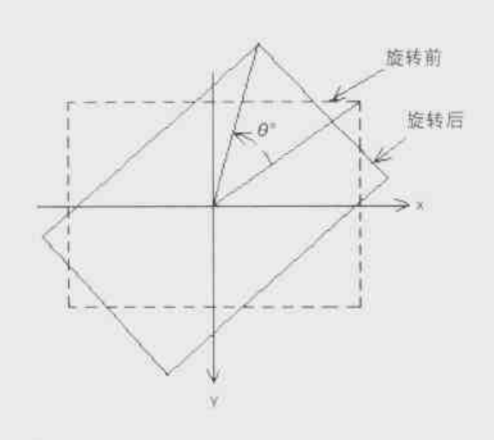


图 10-11



图 10-12

10.5　复杂变形

组合上述的放大、缩小、平移和旋转操作，就可以实现各种各样的变形。到目前为止，所说明的方法都是以原点为中心进行的变形，可是以任意点为中心旋转、放大缩小也是可能的。例如，以（x_0，y_0）为中心旋转，如图 10–13 所示，首先平移（$-x_0$，$-y_0$），使（x_0，y_0）回到原点后，旋转 $\theta°$ 角，最后再平移（x_0，y_0）就可以了。用以下的子程序就能够实现。

```
Shift(image_in, image_out, xsize, ysize, -x0, -y0);
Rotation(image_out, image_in, xsize, ysize, θ);
Shift(image_in, image_out, xsize, ysize, x0, y0);
```

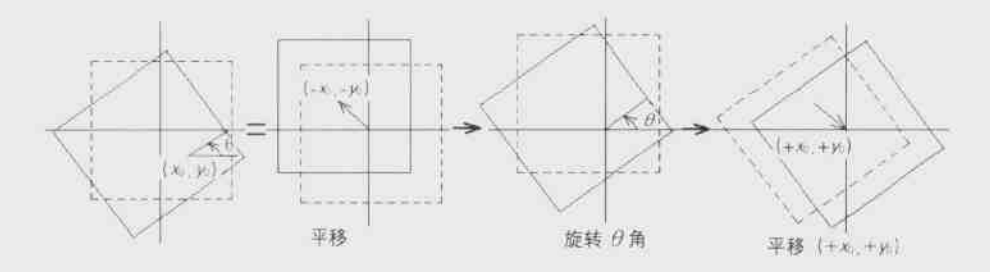


图 10-13

使用这种方法时，在处理过程中，为了计算像素的灰度值，需要不断地计算地址和存取像素，所以要耗费许多时间。为了节省时间，可以用下式先集中计算地址：



$$X = (x - x_0)\cos\theta + (y - y_0)\sin\theta + x_0$$
$$Y = -(x - x_0)\sin\theta + (y - y_0)\cos\theta + y_0 \tag{10.8}$$

逆变换公式如下所示：

$$x = (X - x_0)\cos\theta - (Y - y_0)\sin\theta + x_0$$
$$y = (X - x_0)\sin\theta + (Y - y_0)\cos\theta + y_0 \tag{10.9}$$

集中计算完地址后，读取一次像素，即可计算出变换结果的灰度值。一次完成放大缩小、平移、旋转等复杂变换的程序显示在 List 10.6 中。

这种几何变换被称为二维仿射变换（two、dimensional affine transformation）。二维仿射变换的一般表示公式如下：

$$X = ax + by + c$$
$$Y = dx + ey + f \tag{10.10}$$

逆变换公式如下：

$$x = AX + BY + C$$
$$y = DX + EY + F \tag{10.11}$$

式（10.10）和（10.11）虽然参数不同但形式相同。前面所说明的放大缩小公式（10.2）、平移公式（10.4）和（10.5）、旋转公式（10.6）和（10.7）都包含在公式（10.10）和（10.11）中。

公式（10.10）和（10.11）是 1 次多项式，如果使之成为高次多项式，会产生更加复杂的几何变换。

图 10–1 所示的图像是透视变换（perspective transform）的一个处理范例。一般绘画时会对远处的东西描绘得小一些，透视变换也可以生成类似的效果。如图 10–14 所示，从一点（视点）观看一个物体时，物体在成像平面上的投影图像就是透视变换图像。这种透视变换用以下两式来表达：

$$X = (ax + by + c)/(px + qy + r)$$
$$Y = (dx + ey + f)/(px + qy + r) \tag{10.12}$$

逆变换公式如下所示：

$$x = (AX + BY + C)/(PX + QY + R)$$
$$y = (DX + EY + F)/(PX + QY + R) \tag{10.13}$$

正逆变换的形式相同。在此，a、b、c 与 A、B、C 等是变换系数，决定于视点的位置、成像平面的位置以及物体的大小。这些系数用齐次坐标（homogeneous coordinate）的矩阵形式运算可以简单地求出。实现透视变换的程序显示在 List 10.7 中，处理范例如图 10–15 所示，其中所用的参数为 ax=1.5，ay=3.0，px=py=pz=3.0，rz=10，rx=–75，ry=10，v=10，s=5。



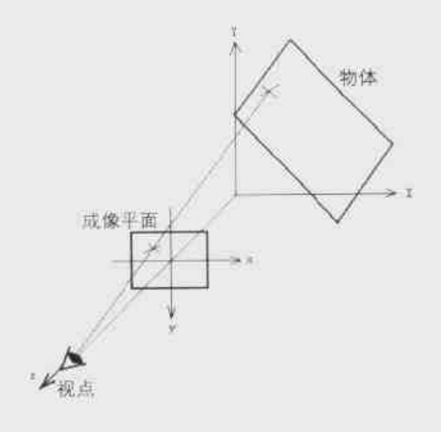


图 10-14



图 10-15

10.6 齐次坐标表示

几何变换采用矩阵处理更为方便。二维平面（x，y）的几何变换能够用二维向量[x，y]和 2×2 矩阵来表现，但是却不能表现平移。因此，为了能够同样地处理平移，增加一个虚拟的维 1，即通常使用三维向量[x，y，1]$'$和 3×3 的矩阵。这个三维空间的坐标（x，y，1）被称为（x，y）的齐次坐标。

基于这个齐次坐标，仿射变换可表现为：

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.14}$$

上式与公式(10.10)是一致的。另外放大缩小表现为：

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.15}$$

平移的齐次坐标表示为：

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.16}$$

旋转的齐次坐标表示为：



$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.17}$$

式（10.15）、式（10.16）、式（10.17）分别与前述的式（10.1）、式（10.4）、式（10.6）一致。组合这些矩阵能够表示各种各样的仿射变换。例如，以（x_0，y_0）为中心旋转，可以表示为如下所示的平移和放大缩小矩阵乘积的形式：

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{10.18}$$

上式展开后与式（10.8）是一致的。

透视变换等是三维空间的变换，用 4 维向量和 4×4 的矩阵来表现。如空间中一点分别在两个坐标系的坐标为（X，Y，Z）和（x，y，z），则其坐标变换公式用旋转矩阵 R 和平移矩阵 t 可描述为：

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{10.19}$$

其中 R 为 3×3 的旋转矩阵（rotation matrix）；t 为三维平移向量（translation vector）；$0=(0,0,0)^T$。这种透视变换经常应用在计算机图形学（computer graphics）等领域。

List 10.1　放大缩小（不正确的做法）

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Scale_NG --- 错误的放大缩小法----------------------
   image_in:    输入图像数据指针
   image_out:   输出图像数据指针
   xsize:       图像宽度
   ysize:       图像高度
   zx:          横向放大率
   zy:          纵向放大率
-----------------------------------------------------------*/
void Scale_NG(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float zx,float zy)
{
 int i, j, m, n;
```



```
    int xs = xsize/2;
    int ys = ysize/2;

    for (j = -ys; j < ys; j++) {
      for (i = -xs; i < xs; i++) {
        m = (int)(zy * j);
        n = (int)(zx * i);
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          *(image_out + (m+ys)*xsize + n+xs) =
          *(image_in + (j+ys)*xsize + i+xs);
      }
    }
}
```

List 10.2 放大缩小（最近邻点法）

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Scale_near---放大缩小（最近邻点法）----------------------------------
    image_in:    输入图像数据指针
    image_out:   输出图像数据指针
    xsize:       图像宽度
    ysize:       图像高度
    zx:          横向放大率
    zy:          纵向放大率
---------------------------------------------------------------------*/
void Scale_near(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float zx,float zy)
{
    int i, j, m, n;
    int xs = xsize/2;
    int ys = ysize/2;

    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++){
        *(image_out + j*xsize + i) = 0;
      }
```



```
    }

    for (j = -ys; j < ys; j++) {
      for (i = -xs; i < xs; i++) {
        if (j > 0) m = (int)(j/zy + 0.5);
        else m = (int)(j/zy - 0.5);
        if (i > 0) n = (int)(i/zx + 0.5);
        else n = (int)(i/zx - 0.5);
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          *(image_out + (j+ys)*xsize + i+xs) =
          *(image_in + (m+ys)*xsize + n+xs);
        else
          *(image_out + (j+ys)*xsize + i+xs) = 0;
      }
    }
}
```

List 10.3 放大缩小（双线性内插法）

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Scale---放大缩小（双线性内插法）------------------------------------------
   image_in:    输入图像数据指针
   image_out:   输出图像数据指针
   xsize:       图像宽度
   ysize:       图像高度
   zx:          横向放大率
   zy:          纵向放大率
-------------------------------------------------------------------------------*/
void Scale(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float zx,float zy)
{
   int i, j, m, n;
   float x, y, p, q;
   int xs = xsize/2;
   int ys = ysize/2;
   int d;
```



```
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++){
        *(image_out + j*xsize + i) = 0;
      }
    }

    for (j = -ys; j< ys; j++) {
      for (i = -xs; i< xs; i++) {
        y = j/zy;
        x = i/zx;
        if (y > 0) m = (int)y;
        else m = (int)(y-1);
        if (x > 0) n = (int)x;
        else n = (int)(x-1);
        q = y - m;
        p = x - n;
        if (q == 1) {q = 0; m = m + 1;}
        if (p == 1) {p = 0; n = n + 1;}
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          d=(int)((1.0-q)*((1.0-p)*(*(image_in+(m+ys)*xsize+n+xs))
                          + p*(*(image_in+(m+ys)*xsize+n+1 +xs)))
                  +q*((1.0-p)*(*(image_in+(m+1+ys)*xsize+n +xs))
                          +p*(*(image_in+(m+1+ys)*xsize+n+1 +xs))));
        else
          d = 0;
        if (d <   0) d =   0;
        if (d > 255) d = 255;
        *(image_out + (j+ys)*xsize + i + xs) = d;
      }
    }
}
```

List 10.4 平移（双线性内插法）

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Shift --- 平移（双线性内插法）-------------------------------------
```



```
   image_in:    输入图像数据指针
   image_out:     输出图像数据指针
   xsize:       图像宽度
   ysize:         图像高度
   px:         横向移动量
   py:         纵向移动量
-------------------------------------------------------------*/
void Shift(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float px,float py)
{
   int i, j, m, n;
   float x, y, p, q;
   int xs = xsize/2;
   int ys = ysize/2;
   int d;

   for (j = -ys; j < ys; j++) {
     for (i = -xs; i < xs; i++) {
       y = j - py;
       x = i - px;
       if (y > 0) m = (int)y;
       else m = (int)(y-1);
       if (x > 0) n = (int)x;
       else n = (int)(x-1);
       q = y - m;
       p = x - n;
       if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
         d=(int)((1.0-q)*((1.0-p)*(*(image_in+(m+ys)*xsize+n+xs))
                         +p*(*(image_in+(m+ys)*xsize+ n+1+xs)))
                 +q*((1.0-p)*(*(image_in+(m+1+ys)*xsize+n +xs))
                      +p*(*(image_in+(m+1+ys)*xsize+ n+1+xs))));
       else
         d = 0;
       if (d <  0) d =  0;
       if (d > 255) d = 255;
       *(image_out + (j+ys)*xsize + i+xs) = d;
     }
   }
```



```
}
```

List 10.5 旋转（双线性内插法）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>

/*--- Rotation ---旋转（双线性内插法）--------------------------------
   image_in:   输入图像数据指针
   image_out:  输出图像数据指针
   xsize:      图像宽度
   ysize:      图像高度
   deg:        回转角
-----------------------------------------------------------------*/
void Rotation(BYTE*image_in,BYTE *image_out,int xsize, int ysize, float deg)
{
   int i, j, m, n;
   float x, y, p, q;
   double  r;
   float c,s;
   int xs = xsize/2;
   int ys = ysize/2;
   int d;

   r = deg*PI/180.0;
   c = (float)cos(r);
   s = (float)sin(r);
   for (j = -ys; j < ys; j++) {
     for (i =  xs; i < xs; i++) {
       y = i*s + j*c;
       x = i*c - j*s;
       if (y > 0) m = (int)y;
       else m = (int)(y-1);
       if (x > 0) n = (int)x;
       else n = (int)(x-1);
       q = y - m;
```



```
        p = x - n;
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          d=(int)((1.0-q)*((1.0-p)*(*(image_in+(m+ys)*xsize+n+xs))
                           +p*(*(image_in+(m+ys)*xsize + n+1+xs)))
                     +q*((1.0-p)*(*(image_in+(m+1+ys)*xsize+n +xs))
                           +p*(*(image_in+(m+1+ys)*xsize+ n+1+xs))));
        else
          d = 0;
        if (d <   0) d =   0;
        if (d > 255) d = 255;
        *(image_out + (j+ys)*xsize + i+xs) = d;
      }
    }
  }
```

List 10.6　放大缩小、旋转、平移（双线性内插法）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>

/*--- Affine --- 仿射变换（移动、旋转、放大缩小）---------------------------
   image_in:      输入图像数据指针
   image_out:     输出图像数据指针
   deg:           旋转角
   zx:            横向比例
   zy:            纵向比例
   px:            横向移动量
   py:            纵向移动量
----------------------------------------------------------------*/
void Affine(BYTE *image_in, BYTE *image_out, int xsize, int ysize, float deg,
     float zx, float zy, float px, float py)
{
   int i, j, m, n;
   float x, y, u, v, p, q;
   double  r;
   float c, s;
```



```
    int xs = xsize/2;
    int ys = ysize/2;
    int d;

    r = deg*PI/180.0;
    c = (float)cos(r);
    s = (float)sin(r);
    for (j = -ys; j < ys; j++) {
      for (i = -xs; i < xs; i++) {
        v = j - py;
        u = i - px;
        y = (u*s + v*c) / zy;
        x = (u*c - v*s) / zx;
        if (y > 0) m = (int)y;
        else m = (int)(y-1);
        if (x > 0) n = (int)x;
        else n = (int)(x-1);
        q = y - m;
        p = x - n;
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          d=(int)((1.0-q)*((1.0-p)*(*(image_in+(m+ys)*xsize + n+xs))
                          +p*(*(image_in+(m+ys)*xsize + n+1+xs)))
                  +q*((1.0-p)*(*(image_in+(m+1+ys)*xsize+ n+xs))
                          +p*(*(image_in+(m+1+ys)*xsize+ n+1+xs))));
        else
          d = 0;
        if (d <  0) d =  0;
        if (d > 255) d = 255;
        *(image_out + (j+ys)*xsize + i+xs) = d;
      }
    }
}
```

List 10.7 透视变换（双线性内插法）

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
```



```
void param_pers(int xsize,int ysize,float k[9], float a,float b,float x0,
 float y0, float z0, float z, float x, float y, float t, float s);
void matrix(double l[4][4], double m[4][4], double n[4][4]);

/*--- Perspective --- 透视变换（双线性内插法）---------------------------
   image_in:          输入图像数据指针
   image_out:         输出图像数据指针
   xsize:             图像宽度
   ysize:             图像高度
   ax:                放大率（横向）
   ay:                放大率（纵向）
   px:                移动量（x）
   py:                移动量（y）
   pz:                移动量（z）
   rz:                回转角（z 轴）
   rx:                回转角（x 轴）
   ry:                回转角（y 轴）
   v:                 视点位置（z）
   s:                 屏幕位置（z）
-----------------------------------------------------------------------*/
void Perspective(BYTE*image_in,BYTE*image_out,int xsize,int ysize,float ax,float ay,
 float px, float py, float pz, float rz,
 float rx, float ry, float v, float s)
{
   int i, j, m, n;
   float x, y, w, p, q;
   float k[9];
   int xs = xsize/2;
   int ys = ysize/2;
   int    d;

   param_pers(xsize, ysize,k,ax,ay,px,py,pz,rz,rx,ry,v,s); //计算变换参数
   for (i = -ys; i < ys; i++) {
     for (j = -xs; j < xs; j++) {
       w = k[0]*j + k[1]*i + k[2];
       x = k[3]*j + k[4]*i + k[5];
       y = k[6]*j + k[7]*i + k[8];
```



```
        x = x/w;
        y = y/w;
        if (y > 0) m = (int)y;
        else m = (int)(y-1);
        if (x > 0) n = (int)x;
        else n = (int)(x-1);
        q = y - m;
        p = x - n;
        if ( (m >= -ys) && (m < ys) && (n >= -xs) && (n < xs) )
          d=(int)((1.0-q)*((1.0-p)*(*(image_in+(m+ys)*xsize+n+xs))
                          +p*(*(image_in+(m+ys)*xsize + n+1+xs)))
                   +q*((1.0-p)*(*(image_in+(m+1+ys)*xsize+n +xs))
                          +p*(*(image_in+(m+1+ys)*xsize+ n+1+xs))));
        else
          d = 0;
        if (d <   0) d =   0;
        if (d > 255) d = 255;
        *(image_out + (i+ys)*xsize + j+xs) = d;
      }
    }
}

/*--- param_pers --- 计算透视变换的参数 --------------------------------
   xsize:     图像宽度
   ysize:     图像高度
   k:         变换参数
   a:         放大率 (x 方向)
   b:         放大率 (y 方向)
   x0:    移动量 (x 方向)
   y0:    移动量 (y 方向)
   z0:    移动量 (z 方向)
   z:         回转角 (z 方向)
   x:         回转角 (x 方向)
   y:         回转角 (y 方向)
   v:         视点位置 (z)
   s:         屏幕位置 (z)
```



```
--------------------------------------------------------------------------*/
void param_pers(int xsize,int ysize, float k[9],float a,float b,float x0,
 float y0, float z0, float z, float x, float y, float t, float s)
{
   double  l[4][4],m[4][4],n[4][4];
   float k1,k2,k3,k4,k5,k6,k7,k8,k9;
   double  u,v,w;
   int xs = xsize/2;
   int ys = ysize/2;

   u=x*PI/180.0; v=y*PI/180.0; w=z*PI/180.0;
   l[0][0]= 1.0/xs;  l[0][1]= 0;      l[0][2]= 0;       l[0][3]= 0;
   l[1][0]= 0;       l[1][1]= -1.0/xs; l[1][2]= 0;      l[1][3]= 0;
   l[2][0]= 0;       l[2][1]= 0;      l[2][2]= 1;       l[2][3]= 0;
   l[3][0]= 0;       l[3][1]= 0;      l[3][2]= 0;       l[3][3]= 1;
   m[0][0]= a;       m[0][1]= 0;      m[0][2]= 0;       m[0][3]= 0;
   m[1][0]= 0;       m[1][1]= b;      m[1][2]= 0;       m[1][3]= 0;
   m[2][0]= 0;       m[2][1]= 0;      m[2][2]= 1;       m[2][3]= 0;
   m[3][0]= 0;       m[3][1]= 0;      m[3][2]= 0;       m[3][3]= 1;
   matrix(l,m,n);  //正规化矩阵乘放大缩小矩阵
   l[0][0]= 1;       l[0][1]= 0;      l[0][2]= 0;       l[0][3]= 0;
   l[1][0]= 0;       l[1][1]= 1;      l[1][2]= 0;       l[1][3]= 0;
   l[2][0]= 0;       l[2][1]= 0;      l[2][2]= 1;       l[2][3]= 0;
   l[3][0]= x0;      l[3][1]= y0;     l[3][2]= z0;      l[3][3]= 1;
   matrix(n,l,m);  //乘移动矩阵
   n[0][0]=  cos(w); n[0][1]= sin(w);  n[0][2]= 0;      n[0][3]= 0;
   n[1][0]= -sin(w); n[1][1]= cos(w);  n[1][2]= 0;      n[1][3]= 0;
   n[2][0]= 0;       n[2][1]= 0;      n[2][2]= 1;       n[2][3]= 0;
   n[3][0]= 0;       n[3][1]= 0;      n[3][2]= 0;       n[3][3]= 1;
   matrix(m,n,l);  // 乘 z 轴旋转矩阵
   m[0][0]= 1;       m[0][1]= 0;      m[0][2]= 0;       m[0][3]= 0;
   m[1][0]= 0;       m[1][1]= cos(u);  m[1][2]= sin(u);  m[1][3]= 0;
   m[2][0]= 0;       m[2][1]= -sin(u); m[2][2]= cos(u);  m[2][3]= 0;
   m[3][0]= 0;       m[3][1]= 0;      m[3][2]= 0;       m[3][3]= 1;
   matrix(l,m,n);  // 乘 x 轴旋转矩阵
   l[0][0]= cos(v);  l[0][1]= 0;      l[0][2]= sin(v);  l[0][3]= 0;
   l[1][0]= 0;       l[1][1]= 1;      l[1][2]= 0;       l[1][3]= 0;
```



```
    l[2][0]= -sin(v); l[2][1]= 0;      l[2][2]= cos(v);  l[2][3]= 0;
    l[3][0]= 0;       l[3][1]= 0;      l[3][2]= 0;       l[3][3]= 1;
    matrix(n,l,m);  //乘 y 轴旋转矩阵
    n[0][0]= 1;       n[0][1]= 0;      n[0][2]= 0;       n[0][3]= 0;
    n[1][0]= 0;       n[1][1]= 1;      n[1][2]= 0;       n[1][3]= 0;
    n[2][0]= 0;       n[2][1]= 0;      n[2][2]= -1;      n[2][3]= 0;
    n[3][0]= 0;       n[3][1]= 0;      n[3][2]= t;       n[3][3]= 1;
    matrix(m,n,l);  // 乘视点坐标变换矩阵
    m[0][0]= 1;       m[0][1]= 0;      m[0][2]= 0;       m[0][3]= 0;
    m[1][0]= 0;       m[1][1]= 1;      m[1][2]= 0;       m[1][3]= 0;
    m[2][0]= 0;       m[2][1]= 0;      m[2][2]= 1/s;     m[2][3]= 1/s;
    m[3][0]= 0;       m[3][1]= 0;      m[3][2]= -1;      m[3][3]= 0;
    matrix(l,m,n);  // 乘透视变换矩阵
    l[0][0]= xs;      l[0][1]= 0;      l[0][2]= 0;       l[0][3]= 0;
    l[1][0]= 0;       l[1][1]= -xs;    l[1][2]= 0;       l[1][3]= 0;
    l[2][0]= 0;       l[2][1]= 0;      l[2][2]= 1;       l[2][3]= 0;
    l[3][0]= 0;       l[3][1]= 0;      l[3][2]= 0;       l[3][3]= 1;
    matrix(n,l,m);  // 乘正规化逆矩阵
    k1=(float)(m[0][3]);  k2=(float)(m[1][3]);  k3=(float)(n[3][3]);
    k4=(float)(m[0][0]);  k5=(float)(m[1][0]);  k6=(float)(n[3][0]);
    k7=(float)(m[0][1]);  k8=(float)(m[1][1]);  k9=(float)(n[3][1]);
    k[0]=k7*k2-k8*k1; k[1]=k5*k1-k4*k2; k[2]=k4*k8-k7*k5;
    k[3]=k8*k3-k9*k2; k[6]=k9*k1-k7*k3; k[4]=k6*k2-k5*k3;
    k[7]=k4*k3-k6*k1; k[5]=k5*k9-k8*k6; k[8]=k7*k6-k4*k9;
}

/*--- matrix --- 矩阵计算 ------------------------------------------
    l:  输入矩阵 1
    m:  输入矩阵 2
    n:  输出矩阵
-----------------------------------------------------------------*/
void matrix(double l[4][4], double m[4][4], double n[4][4])
{
    int   i, j, k;
    double  p;

    for (i = 0; i < 4; i++) {
```



```
    for (j = 0; j < 4; j++) {
      p = 0;
      for (k = 0; k < 4; k++) p = p + l[i][k]*m[k][j];
      n[i][j] = p;
    }
  }
}
```





图 8-6

10 如果在窗体中选择的表为 Orders 和该表的所有列，单击“选择要在窗体中显示的表和列”对话框中的“下一步”，则显示“选择显示式样”对话框，如图 8-7 所示。在如何显示数据对话框中，有两种显示数据的方式。

- 一种是用网格显示，这使用户能够一次得到多行数据。
- 一种是用单独的控件显示。这个选项允许用户多次以单记录的方式操作数据表。向导可以产生一个导航控件，它可以让用户在记录之间游动。另外向导还可以建立添加、删除、取消 3 个按钮。

数据窗体向导
选择显示样式
可以一次显示单个记录，也可以同时显示所有记录。
您希望如何显示数据?
网格中的所有记录(G)
单个控件中的单个记录(S)
在该窗体中您还需要哪些附加控件?
全部取消 – 取消对数据集中所有记录所做的更改(C)
如果选择单个控件，则可以添加用于编辑和导航的控件：
向导目前已拥有其所需的信息。单击“完成”退出并生成新窗体。
取消
< 上一步(B)
完成(F)

图 8-7

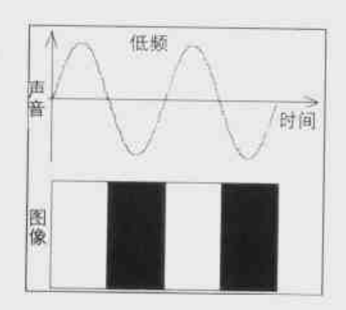


11.1 频率的世界

本章的主题与以前所介绍的图像处理方法和视点完全不同。前面介绍了许多图像处理方法，无论哪一种都是在视觉上容易被理解的方法。这是因为它们都利用了图像的视觉性质。然而，所谓的频率（frequency）听起来似乎与图像无关。

说起频率，人们自然会联想到普通的声音世界。因此，可以把图像的频率用声音来类推说明。通过图 11-1 可清楚地看出图像的低频（low frequencies）代表粗略部分，即总体灰度的平滑区域。图像的高频（high frequencies）代表细微部分，即边缘和噪声，其中，（a）是粗略部分，（b）是细微部分。那么，用频率来处理是为了达到什么目标呢？还是用声音作比较来说明，声音的频率处理应该是我们平常经历过的，例如，通过立体声音响设备所附带的音调控制器，把 TREBLE（高音）调低的话将发出很闷的声音，相反把 BASS（低音）调低的话将发出尖利的声音。图像也是同样可以进行频率处理的。以图 11-2 的处理为例，去掉高频成分的话，细微部分就消失了，从而图像变得模糊不清。相反去掉低频成分的话，粗略部分不见了，仅留下边缘，其中，（a）是原始图像，（b）是去掉高频，（c）是去掉低频。

(a) | (b)



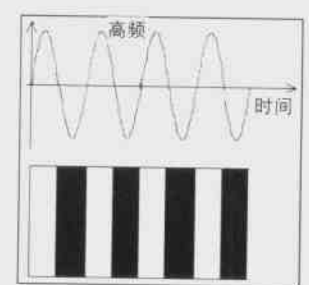


图 11-1

(a)|(b)|(c)







图 11-2



用频率来处理图像，首先需要把图像进行频率变换（即 frequency domain，频率域）。这种变换是使用傅立叶变换（Fourier transform）的方法进行的，而频率处理本身就是一个研究领域。

这本书的宗旨是浅显易懂地进行解说，而对于频率，只罗列程序和处理示例，其中也会包括相当多的在某种程度上详细说明的内容。

首先介绍把一维信号变换到频率域的方法，接着说明图像的二维信号的频率变换内容。下面开始讲解基于频率的图像处理。

11.2 频率变换

任意波形都能够表现为单纯的正弦波的和，这是频率变换的基础。例如，图 11-3(a)所示的波形能够分解成图 11-3(b)、图 11-3(c)、图 11-3(d)、图 11-3(e)所示的四个具有不同频率的正弦波。

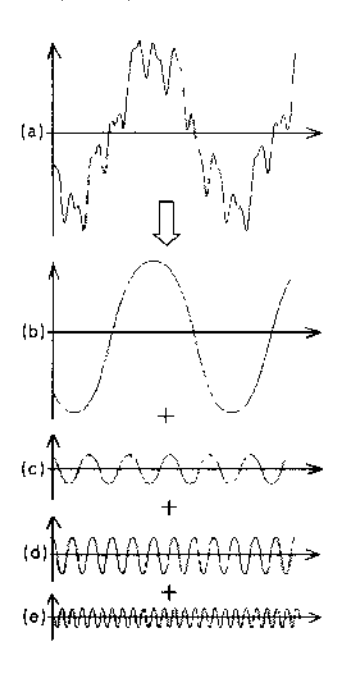


图 11-3

那么，取出图 11-3(d)所示的波形详细分析一下。如图 11-4 所示，用虚线表示振幅为 1 且通过原点的基本正弦波，实线波可以由幅度（amplitude）A 与相位（phase）ϕ 确定。从而图



11−3(b)、图 11−3(c)、图 11−3(d)、图 11−3(e)的四个波形，可画成水平轴为频率 f、垂直轴为幅度 A 的图形，以及水平轴为频率 f、垂直轴为相位 ϕ 的图形，如图 11−5 所示。这种反映频率与幅度、相位之间关系的图形称为傅立叶频谱（Fourier spectrum）。这样，把图 11−3(a) 的波形变换到图 11−5 的频率域中。

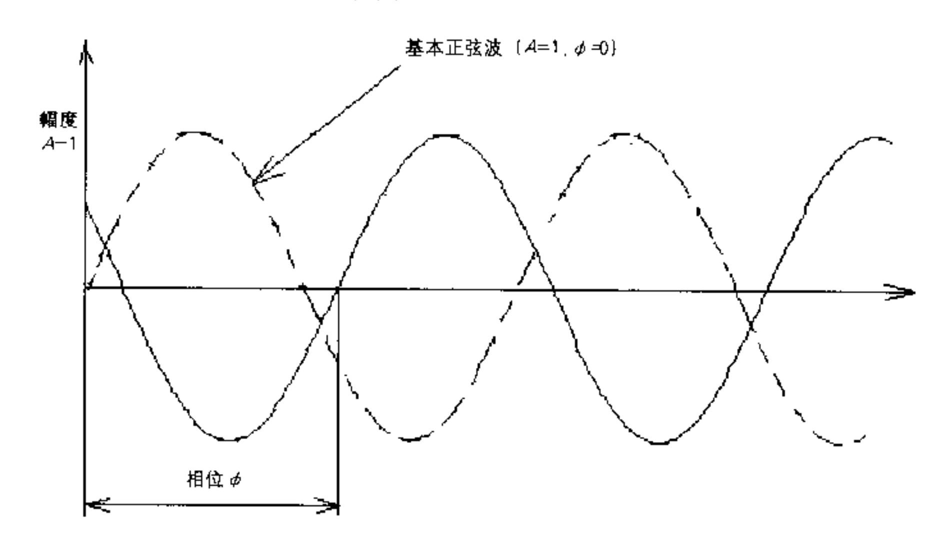


图 11-4

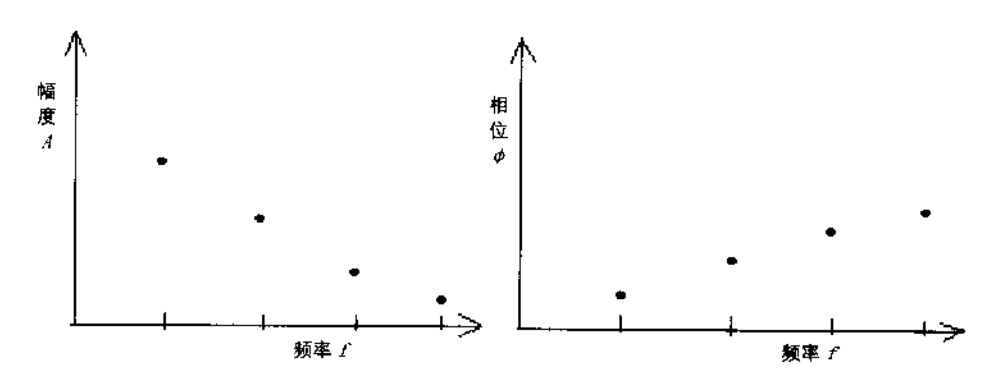


图 11-5

无论在空间域（spatial domain）中多么复杂的波形都可以变换到频率域（frequency domain）中。一般在频率域中也是连续的形式，如图 11−6 所示。



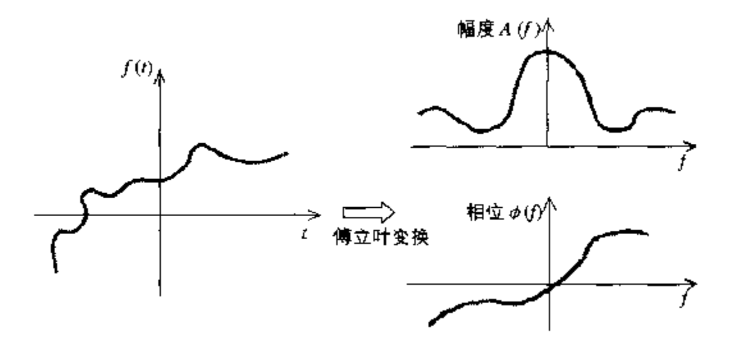


图 11-6

用公式表示为：

$$f(t) \underset{\text{逆傅立叶变换}}{\overset{\text{傅立叶变换}}{\rightleftarrows}} A(f), \phi(f) \tag{11.1}$$

这种变换称为傅立叶变换（Fourier transform），它属于正交变换（orthogonal transform）的一种。

一般在傅立叶变换中为了同时表示幅度 A 和相位 ϕ，可采用复数（complex number）形式。复数是由实数部 a 和虚数部 b 两部分组合表示的数，即用如下公式表示：

$$a + jb \qquad 其中\ (j = \sqrt{-1}) \tag{11.2}$$

采用这个公式就能把幅度和相位这两个概念用一个复数来处理。从而，式（11.1）的傅立叶变换可以使用复函数 $F(f)$ 或者 $F(\omega)$ 表示为：

$$f(t) \underset{\text{逆傅立叶变换}}{\overset{\text{傅立叶变换}}{\rightleftarrows}} F(f)\text{或者 } F(\omega) \tag{11.3}$$

问题是如何从 $f(t)$ 导出 $F(f)$ 或者 $F(\omega)$？这个导出过程在此不介绍，其结果如下式所示：

$$F(f) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ft}dt \qquad 傅立叶变换$$

$$f(t) = \int_{-\infty}^{\infty} F(f)e^{j2\pi ft}dx \qquad 逆傅立叶变换$$

(11.4)

或者

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \qquad 傅立叶变换$$

$$f(t) = \int_{-\infty}^{\infty} F(\omega)e^{j\omega t}dx \qquad 逆傅立叶变换$$

其中，角频率 $\omega=2\pi f$。这就是所有频率处理都要用到的非常重要的基础公式。但是，在此并不深入探讨这个公式，本书的目的是用计算机来处理数字图像。计算机领域与数学领域的不同在于如下两点：一是到目前为止，数学领域所涉及的信号 $f(t)$ 为如图 11−7(a)所示的连续信号（模拟信号），而计算机领域所处理的信号如图 11−7(b)所示，是经采样后的数字信号；



另一点是数学上考虑无穷大是通用的，可是计算机必须进行有限次的运算。上述限制的傅立叶变换称为离散傅立叶变换（discrete Fourier transform，DFT）。

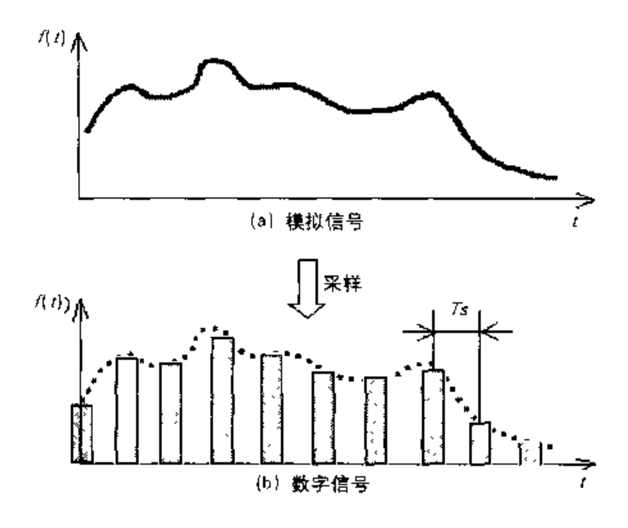


图 11-7

11.3 离散傅立叶变换

离散傅立叶变换（DFT）可以通过把式（11.4）的傅立叶变换参数变为离散值来导出。现假定输入信号为 $x(0)$、$x(1)$、$x(2)$、…、$x(N-1)$共 N个离散值，那么变换到频率域的结果（复数）如图 11-8 所示，也是 N个离散值 $X(0)$、$X(1)$、$X(2)$、…、$X(N-1)$。

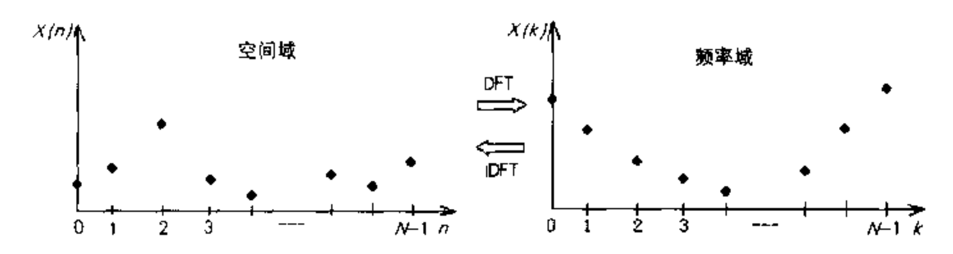


图 11-8

其关系式如下所示：



$$X(k)=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n)W^{kn} \qquad (DFT)$$
$$x(n)=\frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}X(k)W^{-kn} \qquad (IDFT) \tag{11.5}$$

其中，$k=0, 1, 2, \cdots, N-1$；$n=0, 1, 2, \cdots, N-1$；$W=e^{-j\frac{2\pi}{N}}$；IDFT 为逆离散傅立叶变换（inverse discrete Fourier Transform）。这就是DFT 的基本运算公式。积分运算被求和运算所代替，W被称为旋转算子。

在复数领域有欧拉公式，如图11-9 和式（11.6）所示。

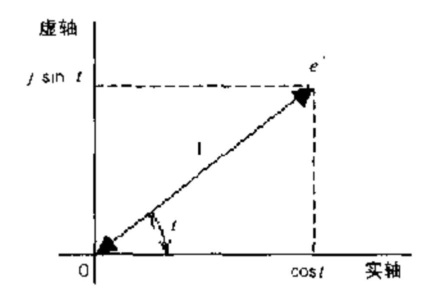


图 11-9

$$e^{jt}=\cos t+j\sin t \tag{11.6}$$

旋转算子可以用欧拉公式置换如下：

$$W^{kn}=e^{-j\frac{2\pi}{N}kn}=\cos\left(\frac{2\pi}{N}kn\right)-j\sin\left(\frac{2\pi}{N}kn\right) \tag{11.7}$$

把式（11.7）代入式（11.5），就只有三角函数和求和运算了，从而能够用计算机进行计算。但是其计算量相当大。因此人们提出了快速傅立叶变换（fast Fourier transform，FFT）的算法，当数据是 2 的正整数次方时，可以节省相当大的计算量。

在此不对算法进行详细说明，只在 List 11.1 中列出进行 FFT 运算的程序。有了这个程序就能够往返于空间域和频率域了。在这个函数 fft1 中将信号向频率域变换时在 a_rl[]中输入信号值，而在 a_im 中都输入 0。信号归根结底都是实数，虚部是不存在的。由于结果是复数，实部 a_rl[]和虚部 a_im[]将分别被计算出来。如果想要了解幅度特性 A（amplitude characteristic）和相位特性 ϕ（phase characteristic）的话，可进行如下变换：



$$A = \sqrt{a_rl^2 + a_im^2}$$
$$\phi = \tan^{-1}\left(\frac{a_im}{a_rl}\right) \tag{11.8}$$

这样所得到的 N 个数列都是什么频率分量呢？如图 11—10 所示，实际上，最左边的为直流分量，最右边为采样频率分量。另外还有一个突出的特点，就是以采样频率的 1/2 处的点为中心，幅度特性左右对称，相位特性中心点对称。

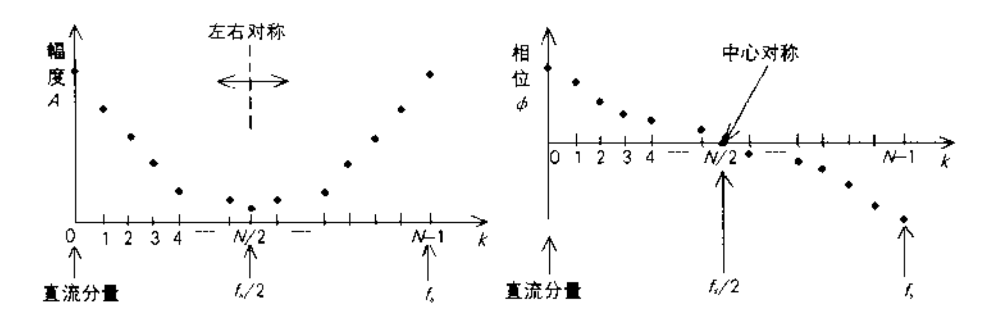


图 11-10

首先来了解采样频率（sampling frequency）和采样定理（sampling theorem）的概念。如图 11—7 所示，由某时间间隔 T 秒对模拟图像进行采样后得到数字图像，这时称 $1/T$（Hz）为采样频率。根据采样定理，数字信号最多只能表示采样频率的 1/2 的模拟信号。例如，CD 采用 44.1 kHz 采样频率，理论上只能表示 0～22.05 kHz 的声音信号。因此，当采样频率为 f_s 时，模拟信号用数字信号置换的含义实质上就只具有 0～ $f_s/2$ 之间的值。

11.4　图像的二维傅立叶变换

从这节开始进入正题，到目前为止，所介绍的所有信号都是一维信号，由于图像是平面的，所以是二维信号，具有水平和垂直两个方向上的频率。另外在图像的频谱中常常把频率平面的中心作为直流分量。在 List 11.1 中令 OPT=1 即为这种形式。

图 11—11 是当水平频率为 u、垂直频率为 v 时与实际图像对应的情形。另外，同样二维频谱的幅度特性是以幅度 A 轴为对称轴的对称，相位特性是以原点为对称点的点对称。

计算二维频率，比较简单的方法是：分别进行水平方向的一维 FFT 和垂直方向的一维 FFT。二维 FFT 的程序被表示在 List 11.2 中。在 List 11.2 中按照图 11—12 所示的处理框图来实现二维 FFT。



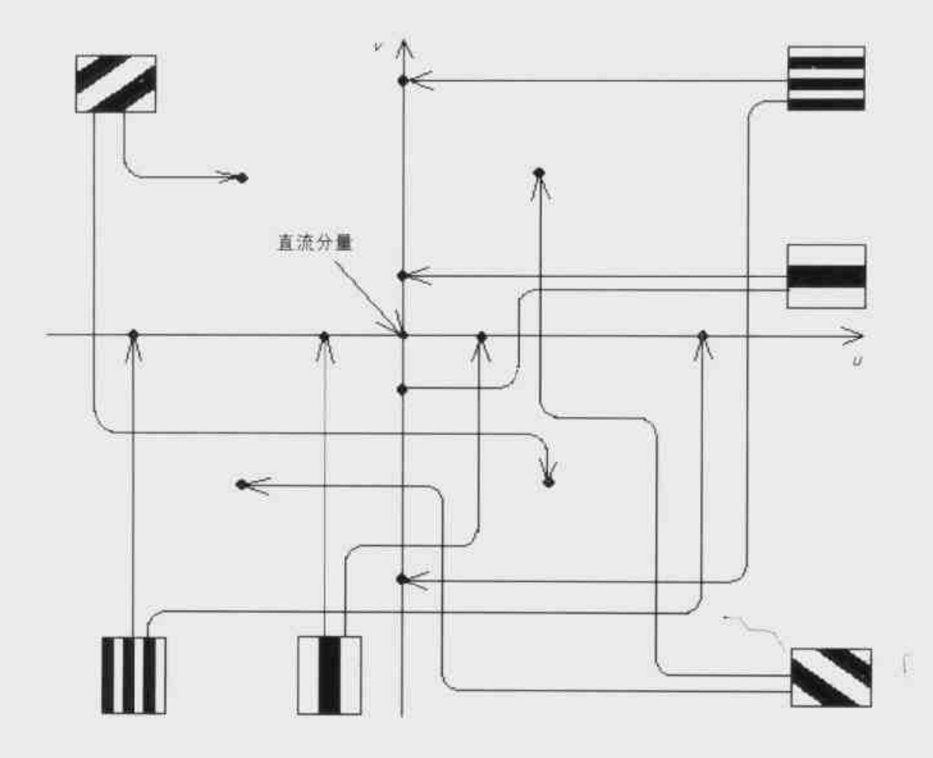


图 11-11

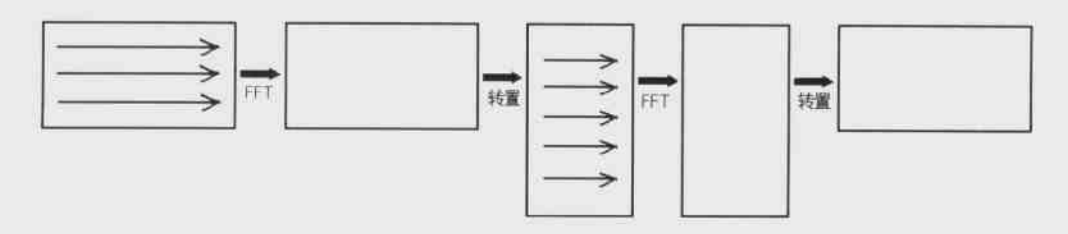


图 11-12

使用 List 11.3 的程序对实际图像进行二维 FFT。在此把幅度特性作为灰度值来图像化，结果如图 11-13 所示。图 11-13(a)与图 11-13(b)进行比较，(a) 是细节少的图像，(b) 是细节多的图像，可以发现，细节少的图像上低频较多，而细节多的图像上高频较多。



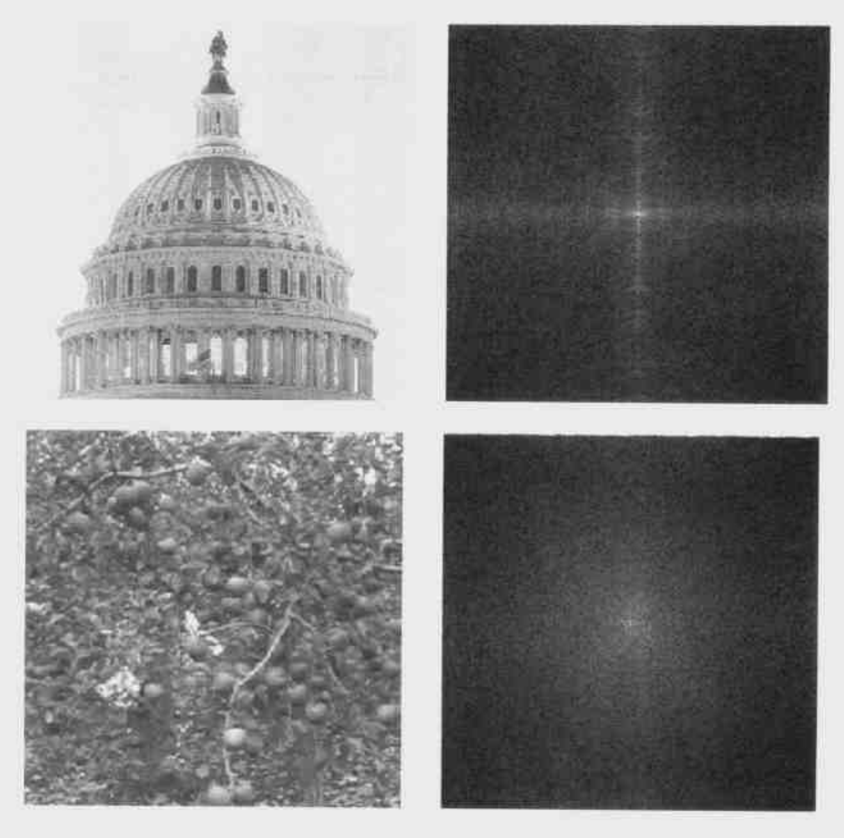



图 11-13

11.5 滤波处理

滤波器（filter）的作用就是使某些东西通过，某些东西阻断。频率域中的滤波器则是使某些频率通过，使某些频率被阻断。

下面使用 List 11.4 的程序实际处理一下。这个程序通过设定参数 *a* 和 *b* 的值，使 *a* 以上，*b* 以下的频率（斜线表示的频率）通过，其他的频率被阻断，以此来进行滤波处理，如图 11–14 所示。图 11–15 所示是把图像经 DFT 处理得到频率成分的高频分量设置为 0，再进行 IDFT 处理变换回图像的处理，程序显示在 List 11.5 中。从处理结果可以看出，图像的高频分量（细节部分）确实消失了，从而变模糊了。下面再看一下把低频分量设置为 0，其处理结果如图 11–16 所示。结果边缘被提取出来了，这是由于许多高频分量包含在边缘中的原因。



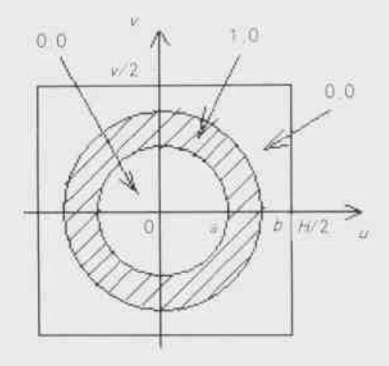


图 11-14

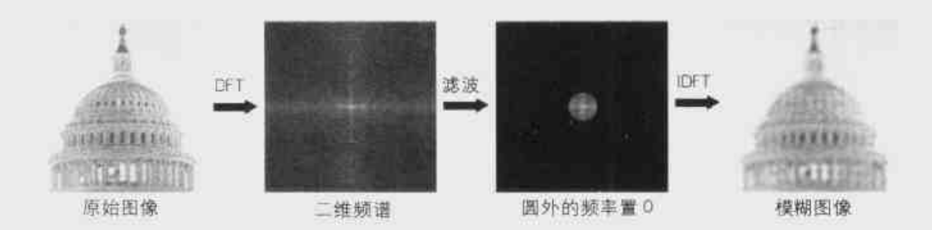


图 11-15

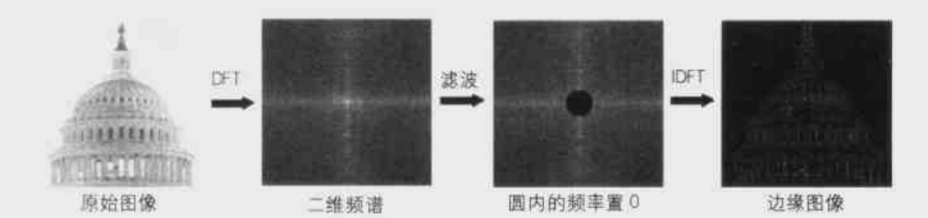


图 11-16

滤波处理可以理解为滤波器的频率和图像的频率相乘，实际上变更这个滤波器的频率特性可以得到各种各样的处理。下面用数学公式整理一下。假定输入图像为 $f(i, j)$，则图像的频率 $F(u, v)$ 变为：

$$F(u,v) = D[f(i,j)] \tag{11.9}$$

其中 $D[\ ]$ 表示 DFT，如果滤波器的频率特性表示为 $S(u, v)$，则处理图像 $g(i, j)$ 表示为：

$$g(i,j) = D^{-1}[F(u,v) \cdot S(u,v)] \tag{11.10}$$

其中 $D^{-1}[\ ]$ 表示 IDFT 在此，假定 $S(u, v)$ 经 IDFT 得到 $s(i, j)$，那么式（11.10）将变形为：



$$
\begin{aligned}
g(i,j) &= D^{-1}[F(u,v)\cdot S(u,v)] \\
&= D^{-1}[F(u,v)]\otimes D^{-1}[S(u,v)] \\
&= f(i,j)\otimes s(i,j)
\end{aligned}
\tag{11.11}
$$

其中 ⊗ 符号被称为卷积运算（convolution），卷积运算在以前的图像处理中曾经出现过多次了。在第 4 章的轮廓提取部分中出现的算子，利用微分算子进行的微分运算就是卷积运算，例如拉普拉斯算子。从式（11.11）可以得到下面非常重要的性质：在图像上（空间域）的卷积运算与频率域的乘积运算是完全相同的操作。从这个结果可见，拉普拉斯算子实际上是让图像的高频分量通过的滤波处理，从而增强了高频成分。同样在第 5 章中所述的平滑化（移动平均法），是让低频分量通过的滤波处理。

频率域处理主要在图像复原（image restoration）（劣化图像如何恢复成原始图像状态）等领域中使用。另外在图像传输（image transmission）（如何在有限的容量下可靠地传输大量的图像信息）等领域中应用得也比较多。

List 11.1　一维FFT

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define OPT   1 // OPT = 1 光学 DFT(直流成分在中间)
                // OPT = 0 一般 DFT(直流成分在左端)

void fft1core(float a_rl[], float a_im[], int length,
 int ex, float sin_tbl[], float cos_tbl[], float buf[]);
void cstb(int length, int inv, float sin_tbl[], float cos_tbl[]);
void birv(float a[], int length, int ex, float b[]);

/*--- FFT1 --- 1次傅立叶变换 -----------------------------------------
    a_rl:  实部（输入输出）
    a_im:  虚部（输入输出）
    ex:      数据个数 = 2 的 ex 次方
    inv:     1: DFT  -1: 逆 DFT
--------------------------------------------------------------*/
int FFT1(float a_rl[], float a_im[], int ex, int inv)
{
```



```
    int   i, length = 1;
    float   *sin_tbl;    //Sin 数据配列
    float   *cos_tbl;    //Cos 数据配列
    float *buf;          //工作用数列

    for (i = 0; i < ex; i++) length *= 2;   //计算数据个数
    sin_tbl = (float *)malloc((size_t)length*sizeof(float));
    cos_tbl = (float *)malloc((size_t)length*sizeof(float));
    buf = (float *)malloc((size_t)length*sizeof(float));
    if ((sin_tbl == NULL) || (cos_tbl == NULL) || (buf == NULL)) {
      return -1;
    }
    cstb(length, inv, sin_tbl, cos_tbl);  //计算 Sin、Cos 数据
    fft1core(a_rl, a_im, length, ex, sin_tbl, cos_tbl, buf);
    free(sin_tbl);
    free(cos_tbl);
    return 0;
}

/*--- fft1core --- 一次傅立叶变换的主计算部分 --------------------------
    a_rl:      数据实数部分（输入输出）
    a_im:      数据虚数部分（输入输出）
    ex:        数据个数 = 2 的 ex 次方
    sin_tbl:  //Sin 数据配列
    cos_tbl:  //Cos 数据配列
-----------------------------------------------------------------*/
void fft1core(float a_rl[], float a_im[], int length,
    int ex, float sin_tbl[], float cos_tbl[], float buf[])
{
    int   i, j, k, w, j1, j2;
    int   numb, lenb, timb;
    float  xr, xi, yr, yi, nrml;

    if (OPT == 1) {
      for (i = 1; i < length; i+=2) {
        a_rl[i] = -a_rl[i];
        a_im[i] = -a_im[i];
```



```
    }
  }
  numb = 1;
  lenb = length;
  for (i = 0; i < ex; i++) {
    lenb /= 2;
    timb = 0;
    for (j = 0; j < numb; j++) {
      w = 0;
      for (k = 0; k < lenb; k++) {
        j1 = timb + k;
        j2 = j1 + lenb;
        xr = a_rl[j1];
        xi = a_im[j1];
        yr = a_rl[j2];
        yi = a_im[j2];
        a_rl[j1] = xr + yr;
        a_im[j1] = xi + yi;
        xr = xr - yr;
        xi = xi - yi;
        a_rl[j2] = xr*cos_tbl[w] - xi*sin_tbl[w];
        a_im[j2] = xr*sin_tbl[w] + xi*cos_tbl[w];
        w += numb;
      }
      timb += (2*lenb);
    }
    numb *= 2;
  }
  birv(a_rl, length, ex, buf);    //实数数据的排列
  birv(a_im, length, ex, buf);    //虚数数据的排列
  if (OPT == 1) {
    for (i = 1; i < length; i+=2) {
      a_rl[i] = -a_rl[i];
      a_im[i] = -a_im[i];
    }
  }
  nrml = (float)(1.0 / sqrt((float)length));
```



```
  for (i = 0; i < length; i++) {
    a_rl[i] *  nrml;
    a im[i] *= nrm.;
  }
}

/*--- cstb --- 计算 Sin、Cos 数据  ------------------------------------------
  length:       数据个数
  inv:          1: DFT, -1: 逆 DFT
  sin_tbl:      //Sin 数据配列
  cos_tbl:      //Cos 数据配列
-------------- --   . --------------------------------------------*/
void cstb(int length, int inv, float sin tbl[], float cos tbl[])
{
  int   i;
  float   xx, arg;

  xx = (float)(((-PI) * 2.0) / (float)length);
  if (inv < 0) xx = -xx;
  for (i = 0; i < length; i++) {
    arg = (float)i * xx;
    sin_tbl[i] = (float)sin(arg);
    cos_tbl[i] = (float)cos(arg);
  }
}

/*--- birv --- 排列数据 ---------------------------------------------
  a:      数据配列
  length: 数据个数
  ex:     数据个数 = 2 的 ex 次方
  b:      工作用配列
-----------------------------------------------------------------*/
void birv(float a[], int length, int ex, float b[])
{
  int i, ii, k, bit;

  for (i = 0; i < length; i++) {
```



```
      for (k = 0, ii=i, bit=0; ; bit<<=1, ii>>=1) {
        bit = (ii & 1) | bit;
        if (++k == ex) break;
      }
      b[i] = a[bit];
    }
    for (i = 0; i < length; i++)
      a[i] = b[i];
  }
```

List 11.2 二维FFT

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void cstb(int length, int inv, float sin_tbl[], float cos_tbl[]);
void rvmtx1(float *a, float *b, int xsize, int ysize);
void rvmtx2(float *a, float *b, int xsize, int ysize);

/*    FFT2 ---二次傅立叶变换 ----------------------------------------
   (仅限于 xsize、 ysize 是 2 的次方)
  a_rl:      数据的实数部分
  a_im:      数据的虚数部分
  inv:       1: DFT  -1: 逆 DFT
  xsize:     数据宽度
  ysize:     数据长度
-------------------------------------------------------------------*/
int FFT2 (float *a_rl, float *a_im, int inv, int xsize, int ysize)
{
  float *b_rl;       //数据转置作业用配列 (实数部)
  float *b_im;       //数据转置作业用配列 (虚数部)
  float *hsin_tbl;   //计算水平 Sin 用配列
  float *hcos_tbl;   //计算水平 Cos 用配列
  float *vsin_tbl;   //计算垂直 Sin 用配列
  float *vcos_tbl;   //计算垂直 Cos 用配列
```



```
float *buf_x;     //水平方向作业用配列
float *buf_y;     //垂直方向作业用配列
int   i;

b_rl = (float *)calloc((size_t)xsize*ysize, sizeof(float));
b_im = (float *)calloc((size_t)xsize*ysize, sizeof(float));
hsin_tbl = (float *)calloc((size_t)xsize, sizeof(float));
hcos_tbl = (float *)calloc((size_t)xsize, sizeof(float));
vsin_tbl = (float *)calloc((size_t)ysize, sizeof(float));
vcos_tbl = (float *)calloc((size_t)ysize, sizeof(float));
buf_x = (float *)malloc((size_t)xsize*sizeof(float));
buf_y = (float *)malloc((size_t)ysize*sizeof(float));
if ((b_rl == NULL) || (b_im == NULL)
  || (hsin_tbl == NULL) || (hcos_tbl == NULL)
  || (vsin_tbl == NULL) || (vcos_tbl == NULL)
  || (buf_x == NULL) || (buf_y == NULL)) {
  return -1;
}
cstb(xsize, inv, hsin_tbl, hcos_tbl);  //计算水平用 Sin,Cos 配列
cstb(ysize, inv, vsin_tbl, vcos_tbl);  //计算垂直用 Sin,Cos 配列

int x_exp = (int)(log((double)xsize)/log((double)2));
//水平方向的傅立叶变换
for (i = 0; i < ysize; i++) {
  fft1core(&(*(a_rl + (long)i*xsize)), &(*(a_im + (long)i*xsize)),
        xsize, x_exp, hsin_tbl, hcos_tbl, buf_x);
}
//二维数据的倒置
rvmtx1(a_rl, b_rl, xsize, ysize);
rvmtx1(a_im, b_im, xsize, ysize);

//垂直方向的傅立叶变换
int y_exp = (int)(log((double)ysize)/log((double)2));
for (i = 0; i < xsize; i++) {
  fft1core(&(*(b_rl + ysize*i)), &(*(b_im + ysize*i)),
        ysize, y_exp, vsin_tbl, vcos_tbl, buf_y);
}
```



```
//二维数据的倒置
rvmtx2(b_rl, a_rl, xsize, ysize);
rvmtx2(b_im, a_im, xsize, ysize);

free(b_rl);
free(b_im);
free(hsin_tbl);
free(hcos_tbl);
free(vsin_tbl);
free(vcos_tbl);
return 0;
}

/*--- rvmtx1 --- 二维数据的倒置 ------------------------------------------
a:          二维输入数据
b:          二维输出数据
xsize:      水平数据个数
ysize:      垂直数据个数
-------------------------------------------------------------------------*/
void rvmtx1(float *a, float *b,
int xsize, int ysize)
{
int i, j;

for (j = 0; j < ysize; j++) {
  for (i = 0; i < xsize; i++)
  *(b + i*ysize + j) = *(a + j*xsize + i);
}
}

/*--- rvmtx2 --- 二维数据的倒置 ------------------------------------------
  a:        二维输入数据
  b:        二维输出数据
  xsize:    水平数据个数
  ysize:    垂直数据个数
  ----------------------------------------------------------------------*/
  void rvmtx2(float *a, float *b,
```



```
    int xsize, int ysize)
    {
    int i, j;

    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++)
        *(b + j*xsize + i) = *(a + i*ysize + j);
    }
}
```

List 11.3　二维FFT结果图像化

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*--- FFTImage --- 将二次 FFT 的变换结果图像化 ------------------------------
    (仅限于 xsize、ysize 是 2 的次方)
    image_in:     输入图像指针
    image_out:    输出图像指针(FFT)
    xsize:        图像宽度
    ysize:        图像高度
-----------------------------------------------------------------------*/
int FFTImage(BYTE *image_in, BYTE *image_out, int xsize, int ysize)
{
    float   *ar;    //数据实数部 (输入输出)
    float   *ai;    //数据虚数部 (输入输出)
    double  norm, max;
    float data;
    long  i, j;

    ar = (float *)malloc((size_t)ysize*xsize*sizeof(float));
    ai = (float *)malloc((size_t)ysize*xsize*sizeof(float));
    if ((ar == NULL) || (ai == NULL)) return -1;
    //读入原图像，变换二维 FFT 输入数据
    for (j = 0; j < ysize; j++) {
```

```
    for (i = 0; i < xsize; i++) {
      ar[xsize*j + i] = (float)(*(image_in + j*xsize + i));
      ai[xsize*j + i] = 0.0;
    }
  }
  //二维 FFT 变换
  if (FFT2(ar, ai, 1, xsize, ysize) == -1)
    return -1;
  //FFT 结果图像化
  max = 0;
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      norm = ar[xsize*j + i]*ar[xsize*j + i]
           + ai[xsize*j + i]*ai[xsize*j + i];  //计算幅度成分
      if (norm != 0.0) norm = log(norm) / 2.0;
      else norm = 0.0;
      ar[xsize*j + i] = (float)norm;
      if (norm > max) max = norm;
    }
  }
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      ar[xsize*j + i] = (float)(ar[xsize*j + i]*255 / max);
    }
  }
  //FFT 结果变图像
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      data = ar[xsize*j + i];
      if (data > 255) data = 255;
      if (data <   0) data = 0;
      *(image_out + j*xsize + i) = (unsigned char)data;
    }
  }
  free(ar);
  free(ai);
  return 0;
```



```
}
```

List 11.4　基于二维FFT的滤波处理

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*    FFTFilter ---二次 FFT 的滤波处理、滤波后的频率域图像化 ----------------
    (仅限于 xsize、 ysize 是 2 的次方)
    image_in:       输入图像数据指针
    image_out:      输出图像数据指针
    xsize:          图像宽度
    ysize:          图像高度
    a, b:           通过区域 (a 以上 b 以下的区域通过)
                    a=0, b=xsize=ysize 时, 全区域通过
-----------------------------------------------------------------*/
int FFTFilter(BYTE*image_in,BYTE*image_out,int xsize,int ysize,int a,int b)
{
    float *ar;    //数据实数部 (输入输出)
    float *ai;    //数据虚数部 (输入输出)
    float *ff;    //滤波子的空间频率特性
    double  norm, max;
    float data;
    long  i, j, circ;

    ar = (float *)malloc((size_t)ysize*xsize*sizeof(float));
    ai = (float *)malloc((size_t)ysize*xsize*sizeof(float));
    ff = (float *)malloc((size_t)ysize*xsize*sizeof(float));
    if ((ar == NULL) || (ai == NULL) || (ff == NULL)) {
      return -1;
    }
    //读入原图像, 变换二维 FFT 输入数据
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        ar[xsize*j + i] = (float)(*(image_in + j*xsize + i));
```



```
    ai[xsize*j + i] = 0.0;
  }
}
//二维 FFT 变换
if (FFT2(ar, ai, 1, xsize, ysize) == -1)
  return -1;

//FFT 结果图像化
max = 0;
for (j = 0; j < ysize; j++) {
  for (i = 0; i < xsize; i++) {
    norm = ar[xsize*j + i]*ar[xsize*j + i]
         + ai[xsize*j + i]*ai[xsize*j + i];  //计算幅度成分
    if (norm != 0.0) norm = log(norm) / 2.0;
    else norm = 0.0;
    ar[xsize*j + i] = (float)norm;
    if (norm > max) max = norm;
  }
}
for (j = 0; j < ysize; j++) {
  for (i = 0; i < xsize; i++) {
    ar[xsize*j + i] = (float)(ar[xsize*j + i]*255 / max);
  }
}

//做成只通过 a 以上 b 以下成分的滤波子
for (j = 0; j < ysize; j++) {
  for(i = 0; i < xsize; i++) {
    data = (float)((i-xsize/2)*(i-xsize/2)
      + (j-ysize/2)*(j-ysize/2));
    circ = (long)sqrt(data);
    if ((circ >= a) && (circ <= b))
      ff[xsize*j + i] = 1.0;
    else
      ff[xsize*j + i] = 0.0;
  }
}
```



```
    //对原图像的频率成分实施滤波
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        ar[xsize*j + i] *= ff[xsize*j + i];
        ai[xsize*j + i] *= ff[xsize*j + i];
      }
    }

    //将结果变为图像数据
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        data = ar[xsize*j + i];
        if (data > 255) data = 255;
        if (data < 0) data = 0;
        *(image_out + j*xsize + i) = (BYTE)data;
      }
    }
    free(ar);
    free(ai);
    free(ff);
    return 0;
}
```

List 11.5 基于二维FFT的滤波处理、逆傅立叶变换

```
#include "StdAfx.h"
#include "BaseList.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*--- FFTFilterImage --- 图像的二次 FFT 变换、滤波处理、逆傅立叶变换-----------
    (仅限于 xsize、 ysize 是 2 的次方)
    image_in:    输入图像数据指针
    image_out:   输出图像数据指针
    xsize:       图像宽度
    ysize:       图像高度
    a, b:        通过区域 (a 以上 b 以下的区域通过)
```



```
                a=0, b=xsize=ysize 时, 全区域通过
-----------------------------------------------------------------*/
int FFTFilterImage(BYTE*image_in,BYTE*image_out,int xsize,int ysize,int a,int b)
{
  float *ar;    //数据实数部（输入输出）
  float *ai;    //数据虚数部（输入输出）
  float *ff;    //滤波子的空间频率特性
  float data;
  long  i, j, circ;

  ar = (float *)malloc((size_t)ysize*xsize*sizeof(float));
  ai = (float *)malloc((size_t)ysize*xsize*sizeof(float));
  ff = (float *)malloc((size t)ysize*xsize*sizeof(float));
  if ((ar == NULL) || (ai == NULL) || (ff == NULL)) {
    return -1;
  }
  //读入原图像，变换二维 FFT 输入数据
  for (j = 0; j < ysize; j++) {
    for (i = 0; i < xsize; i++) {
      ar[xsize*j + i] = (float)(*(image_in + j*xsize + i));
      ai[xsize*j + i] = 0.0;
    }
  }
  //二维 FFT 变换
  if (FFT2(ar, ai, 1, xsize, ysize) == 1)
    return -1;
  //做成只通过 a 以上 b 以下成分的滤波子
  for (j = 0; j < ysize; j++) {
    for(i = 0; i < xsize; i++) {
      data = (float)((i-xsize/2)*(i-xsize/2)
        + (j-ysize/2)*(j-ysize/2));
      circ = (long)sqrt(data);
      if ((circ >= a) && (circ <= b))
        ff[xsize*j + i] = 1.0;
      else
        ff[xsize*j + i] = 0.0;
    }
```



```
    }
    //对原图像的频率成分实施滤波
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        ar[xsize*j + i] *= ff[xsize*j + i];
        ai[xsize*j + i] *= ff[xsize*j + i];
      }
    }
    //实施逆 FFT 变换，将滤波后的频率成分变回为图像
    if (FFT2(ar, ai, -1, xsize, ysize) == -1)
      return -1;
    //将结果变为图像数据
    for (j = 0; j < ysize; j++) {
      for (i = 0; i < xsize; i++) {
        data = ar[xsize*j + i];
        if (data > 255) data = 255;
        if (data <   0) data = 0;
        *(image_out + j*xsize + i) = (BYTE)data;
      }
    }
    free(ar);
    free(ai);
    free(ff);
    return 0;
}
```



```
i = ComboBox1.SelectedIndex
Dim frm2 As New Form2 ()
frm2.ShowDialog ()
'将 Form2 中选择的 CustomerID 显示在组合框中
ComboBox1.Text = frm2.DataGrid1.Item (frm2.DataGrid1.CurrentCell_
.RowNumber,0)
```

8 在窗体 Form2 的 Load 事件过程中写入代码：

```
M1.custadapter.Fill (M1.custset, "customers")
DataGrid1.DataSource = M1.custset.Tables ("customers")
If i < 0 Then i = 0 '不选择，显示第一条记录
'移动到指定的行
DataGrid1.CurrentRowIndex = i
'选中该行
DataGrid1.Select (i)
```

9 在窗体 Form2 中的"确定"命令按钮 Click 事件过程中写入代码：

```
Me.Close ()
```

9.3 ListView 控件

列表视图控件 ListView 是用于显示带图标的项目列表，可用来创建类似于 Windows 资源管理器右侧窗口风格的程序界面。例如：Windows 系统中的文件浏览器等。

1. ListView 控件的四种显示模式

1 / 大图标（LargeIcon）

大图标视图是以大图标方式显示数据。如果控件较大，则项目中可以显示在多列中，如图 9-8 所示。

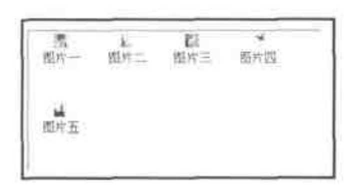


图 9-8

2 / 小图标（SmallIcon）

小图标视图如图 9-9 所示。



12.1 未来的电视电话

随着社会的发展，相隔很远的两个人可以一边看着对方一边通话。现在，既能传送声音又能传送面部图像的电视电话已经登场了。那么，到底是什么样的技术，使得从前只能够传送声音的电话，把数千倍于声音信息的图像传送出去的呢？原因之一就是由于光纤（fiber，光导纤维）等网络的进步，使得大量的数据能够高速地被传送出去，其二就是图像数据压缩技术（image compression）的发展。

本章将通过一些代表性的示例说明图像数据的压缩方法。主要介绍：（1）用于二值图像的"游程长度编码"；（2）用于数据相关性强、连续出现相似数值情况的 DPCM（差分脉冲编码调制）；（3）用于数据不规则情况的"变长编码"，如哈夫曼编码。另外还介绍了静态图像的标准压缩格式 JPEG 和动态图像的标准压缩格式 MPEG。

12.2 无损编码与有损编码

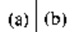
使用公共电话线把数据传送到远处、把数据存储在磁盘上时，需要把这些数据变换成适当的形式（code，码字），这种变换被称为编码（coding）。暗号也是编码的一种，暗号是把原始信息变换成其他人不明白的形式。反过来从编码后的码字返回到原始数据形式的变换称为解码（decoding）。

数据压缩编码大体可分为解码后能够完全恢复原始数据与不能够完全恢复原始数据两种情况。前者的编码称为无损编码（error—free compression，无损压缩），后者称为有损编码（lossy compression，有损压缩）。

无损编码主要是利用数据的统计特性来压缩的，如某值非常多、持续出现相同的数值等特性。基于这些特性，采用后面叙述的游程长度编码、DPCM、变长编码等进行编码时，数据会在没有失真的前提下获得压缩，编码后也可以恢复原始数据。但是 因为利用了数据的统计特性，对于不同特性的数据，有时能进行有效压缩，有时不能进行有效压缩。

与此相对，有损编码是除去一部分数据信息进行压缩，编码后的数据不能被完全地恢复。例如，如图 12—1（a）所示，利用一定间隔的间苗方式来减少图像数据，或者如图 12—1（b）所示，以降低灰度级（比特数）的方式来减少图像数据。然而，采用这些简单的减少数据的方法，在解码时会出现图像模糊、物体变形、虚假轮廓、不自然线条等，造成图像质量的下降。为了在尽可能不损害图像质量的前提下减少数据量，需要施行一些技巧。在实际应用中，图像数据的压缩，只是除去那些在视觉上不明显的高频分量信息。





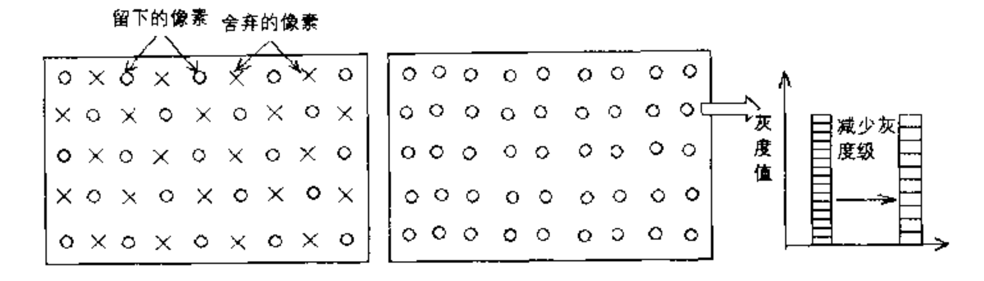


图 12-1

12.3 二值图像编码

首先需要考虑的是只有 0 和 1 两种数值的二值图像的数据压缩方法。这种二值图像的编码方法中，具有代表性的是游程长度编码，或者称为游程编码、行程编码（run–length coding）。这种编码是把连续出现的 0 或者 1 的个数用一个码字来表示，也就是把连续出现的相同的值汇总，并对其长度进行编码的方法。表 12–1 表示了游程长度编码的一个例子。

表 12-1 游程长度编码示例

数据	码字
1	0 0 0
0 (0×1)	0 0 1
0 0 (0×2)	0 1 0
0 0 0 (0×3)	0 1 1
0 0 0 0 (0×4)	1 0 0
0 0 0 0 0 (0×5)	1 0 1
0 0 0 0 0 0 (0×6)	1 1 0
0 0 0 0 0 0 0 (0×7)	1 1 1

例如，使用这些码字对下面的数据进行编码 00000100000001000000100000001。这个数据是由 29 个 0 和 1 组成的。游程长度编码如下 综合连续出现的 0 数据，合计 8 个值被编码。

00000	|1|	0000000	|1|	000000	|1|	0000000	|1|	（计 29 位）
(0×5)	(1)	(0×7)	(1)	(0×6)	(1)	(0×7)	(1)	
101	000	111	000	110	000	111	000	（计 24 位）

使用表 12–1 所示的游程长度编码，每 3 位表示一个数组，共计 24 位，结果用较少的位数表示了原始数据（29 位）。因此，使用游程长度编码，综合多数数据进行编码能够压缩数据。



通常游程长度编码被应用在传真等设备中。

12.4 预测编码

在多值图像压缩方法中，具有代表性的预测编码（predictive coding）是差分脉冲编码调制（differential pulse coding modulation，DPCM）。该方法是利用相邻的像素间常常也具有相近的值的性质，在 DPCM 中把少许的差异信息一个接一个地传递下去进行编码。

如图 12-2 所示，顺序地扫描图像中的各个像素点，DPCM 是利用邻接像素的值（*A*、*B*、*C*、*D*、*E*等）来预测下面的 *X* 位置的像素的值。DPCM 是对预测值 *X'*与实际值 *X* 间的差△*X*(=*X*−*X'*) 进行编码，这个差值越小，预测值越接近于实际值，越能准确地获取信息。预测中所用的不只是一个像素，也有这种情况，就是把左侧的像素值与正上方的像素值加起来除 2 作为预测值，这也需使用多个像素。

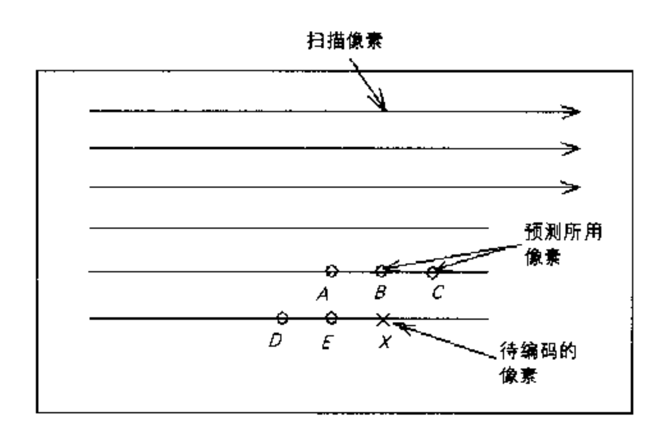


图 12-2

经常使用的预测方法有：

$$X' = E$$
$$X' = (B+E)/2$$
$$X' = B-A+E \qquad (12.1)$$
$$X' = (C+E)/2$$
$$X' = 2E-D$$

其中采用哪种预测方法效果最佳呢？显然根据图像的种类、处理内容的不同而不同，但是如 *X'*=*E* 那样使用前一个（左侧）像素值的方法也是十分有效的。



第一种和第二种的预测方法的 DPCM 编码程序显示在 List 12.1 中，DPCM 的数据直方图以及直方图的百分比计算分别显示在 List 12.2 和 List 12.3 中。DPCM 数据的解码程序显示在 List 12.4 中。由于当预测所用的点在画面外时，该像素将不能使用，这个像素的对应值采用 128。另外，这个 DPCM 程序是以线为单位进行处理的。

12.5 变长编码

DPCM 是对预测值和实际值的差进行编码。这样，本来像素值是在 0～255 范围内的，经过 DPCM 后，却变成了–255～255 的范围内，即像素值范围增加了一倍。因此，为了表示 DPCM 结果的码字，必须准备加倍的数值，用 DPCM 不仅没有使数据量减少，反而增加了。然而，经过 DPCM 后，数据接近 0 的值的概率非常高。如 DPCM 处理结果的数据中，有经常出现的值和不经常出现的值时，给经常出现的值分配较短的码字，就能够减少整体数据量。这种根据数据的值改变码字长度的方法称为变长编码（variable–length coding）。其中哈夫曼编码（Huffman coding）是代表性的变长编码之一。哈夫曼编码是计算各数据出现的概率，从出现概率高的开始依次分配短的码字，所以对出现概率集中的情况比较有效。

哈夫曼编码的例子显示在图 12–3 中。这个哈夫曼编码是 2 位的数据 "00"、"01"、"10"、"11" 的四种类型的值，它们分别以 25/36、6/36、4/36、1/36 的概率出现。由此图可见，出现概率越高所分配的码字则越短。求经过哈夫曼编码后的数据的平均码长，由于各个码字的出现概率已知，所以平均码长为：

$$1\times 25/36+2\times 6/36+3\times 4/36+3\times 1/36=52/36=1.44[\text{bit}]$$

使用哈夫曼编码，数据可以压缩到 70%的程度。

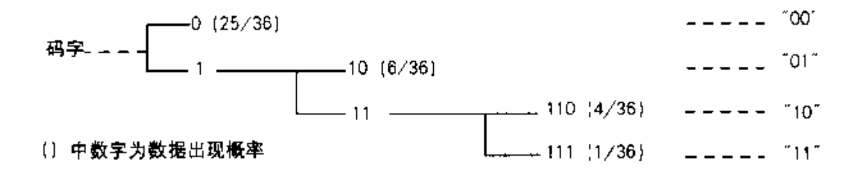


图 12-3

下面对图 12–4 所示的数据进行哈夫曼编码。用 2 位表示的话，14 个数据则合计需要 28 位。与此相比哈夫曼编码则只需要 20 位就可以了。

变长码字如表 12–2 所示，有像 "0" 那样用比原来的 2 位变短了的码字，也有像 "2" 和 "3" 那样用变长了的码字。这样，如果不管数据的统计特性，而使用变长编码，与定长位数



表示相比会出现整体数据量增大的情况。如图 12–5 所示数据，用表 12–2 的码字进行编码，则需要 32 位，比单纯 2 位编码数据量增大了。

数据	0	0	1	0	2	0	0	0	1	0	0	3	0	0	14 个数据
2 位表示	00	00	01	00	10	00	00	00	01	00	00	11	00	00	28 位
哈夫曼编码	0	0	10	0	110	0	0	0	10	0	0	111	0	0	20 位

图 12-4

表 12-2　哈夫曼编码例

数　据		概　率	码　字	码字长（位）
0	"00"	25/36	0	1
1	"01"	6/36	10	2
2	"10"	4/36	110	3
3	"11"	1/36	111	3

数据	0	3	1	0	2	1	3	3	1	2	2	3	1	0	14 个数据
2 位表示	00	11	01	00	10	01	11	11	01	10	10	11	01	00	28 位
哈夫曼编码	0	111	10	0	110	10	111	111	10	110	110	111	10	0	32 位

图 12-5

哈夫曼编码是由数据的概率分布来确定码字的，所以根据数据不同所用的码字也会有所不同。为了确定哈夫曼编码的码字，首先必须预先计算数据的统计量。

然而，前面讲过，DPCM 的出现概率通常集中在 0 附近，因此不一定需要像哈夫曼编码那样先求出现概率再确定码字，用定长编码也可以获得较好的压缩效果。在此显示以 4b 或者 2b 为单位，来改变长度的变长码字的例子，如表 12–3 所示。

表 12-3　变长码字例

数　据	码字 1（以 4b 为单位）		码字 2（以 2b 为单位）	
	码　字	码字长	码　字	码字长
0	0000	4	00	2
1	0001	4	01	2
2	0010	4	10	2
3	0011	4	1100	4



续表

数据	码字 1（以 4b 为单位）		码字 2（以 2b 为单位）	
	码字	码字长	码字	码字长
4	0100	4	1101	4
5	0101	4	1110	4
6	0110	4	111100	6
7	0111	4	111101	6
8	1000	4	111110	6
…	…	…	…	…
14	1110	4	1111111110	10
15	11110000	8	111111111100	12
16	11110001	8	111111111101	12
17	11110010	8	111111111110	12
18	11110011	8	11111111111100	14
…	…	…	…	…
29	11111110	8	11111111111111111110	20
30	111111110000	12	1111111111111111111100	22
31	111111110001	12	1111111111111111111101	22
…	…	…	…	…

一般的图像数据是 1 个像素的值用固定的 8b 来表示，这样可以大概地知道数据与数据之间的交接点，但是变长码字中各码字的长度不定，所以必须从码字本身了解各码字有多长，码字和码字之间的交接点在何处等。对于以 4b 为单位的情况，“1111”持续期间，码字也持续，“1111”以外的数据序列出现时，那里就是码字的交接点。对于以 2b 为单位的情况，也是首先每 2b 查看“11”是否出现，“11”以外出现的位置就是码字的交接点。这样，对于变长码字，可以通过顺序地查看码字的内容来找出各码字的交接点的位置。

List 12.5 表示以 4b 为单位的变长编码的程序，List 12.6 表示其解码的程序。另外，在硬盘等媒体中读写时是以字节为单位进行的，所以在编码程序中把整体的数据量用字节来操作，最后余下的位置 1。

12.6 图像压缩的实例

使用 DPCM 和变长编码，对图 12−6 的（a）和（b）两个图像进行压缩处理。这两个图像的像素值分布（直方图）如图 12−7 所示，其中，a——图像 12.6(a)，b——图像 12.6(b)。可见灰度值在比较广的范围内分布，图像模式的不同灰度值分布的状态完全不同。这两个图像使



用 12.4 节所介绍的 $X'=E$ 的预测方法进行 DPCM 后，所得结果的灰度值分布如图 12–8 所示，其中，a——图像 12.6(a)，b——图像 12.6(b)。可见，进行 DPCM 后无论哪种图像的数据分布都很相似。对这个 DPCM 数据的直方图调查发现，0 值最常出现，几乎正负值都是越接近 0 出现概率越高。从接近出现概率高的 0 值的数开始，顺序地给予较短的变长码字，就能够有效地压缩图像。

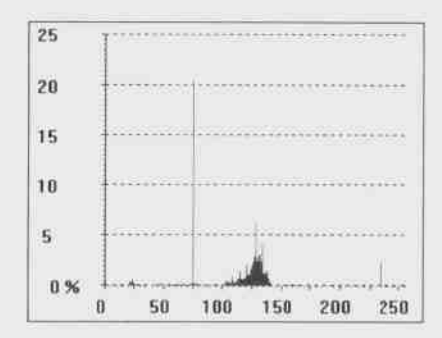
(a) | (b)





图 12-6

(a) | (b)



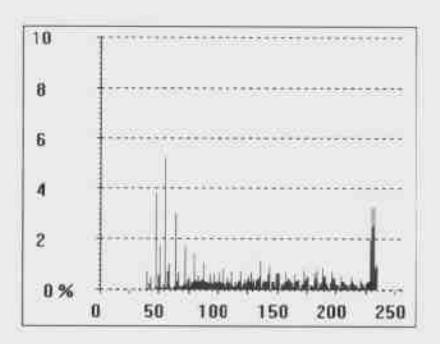
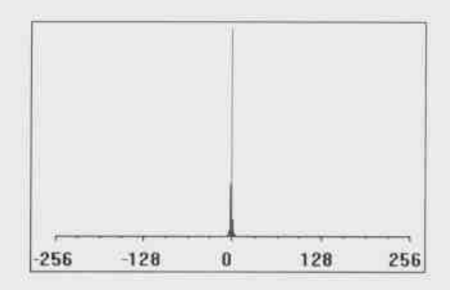


图 12-7



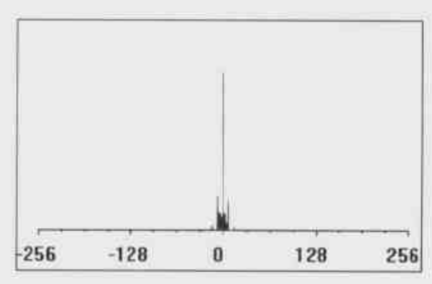


图 12-8



表 12–4 给出了 DPCM 数据 k 和表 12–3 的变长码字值 n 之间的关系。其关系也可以由式（12.2）表示。

表 12-4　DPCM 数据与变长码字值之间的关系

DPCM 数据 k	0	1	−1	2	−2	3	−3	…
变长码字值 n	0	1	2	3	4	5	6	…

$$
\begin{aligned}
n &= 2k-1 \qquad (k>0) \\
n &= -2k \qquad (k \leqslant 0)
\end{aligned}
\tag{12.2}
$$

这个变换程序显示在 List 12.7 中。

综上所述，图像数据的压缩需要经过下列过程：

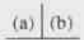
操作步骤

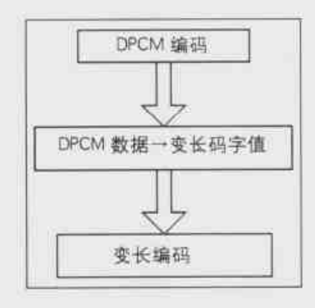
[1] 顺序地扫描图像，在各点求与预测值间的差值，进行 DPCM 编码；

[2] 把 DPCM 数据变换为变长码字值；

[3] 进行变长编码。

List 12.8 列出了图像数据的 DPCM + 变长编码的程序，List 12.9 列出了其解码程序。图像压缩和解码的全过程如图 12–9 所示，其中，(a)是编码，(b)是解码。





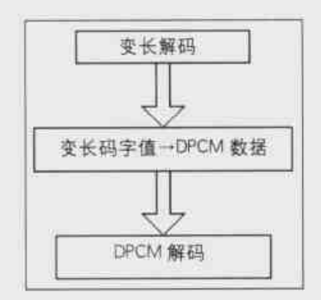


图 12-9

这个编码程序显示在 List 12.8 中。该程序采用的是逐行对图像进行 DPCM 和变长编码的方式。对图 12–6（a）和（b）的图像，用该程序执行后，得到的图像数据的压缩比如表 12–5 所示。

表 12-5　DPCM 和变长码字压缩对不同图像的压缩比

12.4 节中的预测方法	图 12-6 (a)	图 12-6 (b)
$X' = E$	60.98%	75.85%
$X' = (B+E)/2$	56.69%	65.05%



比较两幅图像可见，由于图 12−6（b）比图 12−6（a）的背景复杂，图 12−6（b）的压缩率较低。这可以从图 12−8 所示的 DPCM 数据的分布看出，由于图 12−6（b）与图 12−6（a）相比有细小的图案，所以图 12−6（b）的 DPCM 数据分布比图 12−6（a）的分散宽广。

另外，DPCM 从 12.4 节所介绍的 $X'=E$ 预测方法变化到 $X'=(B+E)/2$ 预测方法时，整体上效率提高了。一般在预测中使用较多像素会更加有效。变长码字的变化单位是选择 2b 还是 4b，与数据的分布有关，不能一概而论。分布偏向某个值时，选择 2b 单位较好，分布分散时，选择 4b 单位较有利。

12.7　图像压缩的标准格式

以静态图像作为对象的 JPEG（Joint Photographic Experts Group，联合图片专家组）和以动态图像作为对象的 MPEG（Moving Picture Experts Group，动态图像专家组），是 ISO（International Standardization Organization，国际标准化组织）和 ITU（International、Telecommunication Union，国际电信联盟）联合制定的图像压缩国际标准。

1．JPEG

JPEG 是本章介绍的无损编码 DPCM 和有损编码 DCT（discrete cosine transform，离散余弦变换）的图像压缩格式。无损编码（包括本章介绍的变长编码以及算术编码等）由于无损压缩，不可能得到很高的压缩比。JPEG 主要是应用有损压缩的 DCT，将由 DCT 得到的参数经过本章介绍的 DPCM、游程长度编码，然后通过量化、斜线扫描等技术，实现高压缩比的目的。

JPEG 图像格式广泛应用于印刷、数码相机、互联网等领域。目前国际上正在开发高压缩比、高画质、高性能的 JPEG2000，预计不再采用 DCT 压缩方式，而是采用下一章将要介绍的小波变换的编码方式对图像进行压缩。

2．MPEG

MPEG 有 MPEG1、MPEG2、MPEG4、MPEG7、MPEG21 等多个版本。

MPEG1 处理的是标准图像交换格式（standard interchange format，SIF）或者称为源输入格式（source input format，SIF）的电视。它是为传输速率在 1.5Mbps（mega bit per second）以下的 CD−ROM 和网络制定的标准，用于在 CD−ROM 上存储数字影视和在网络上传输数字影视。它属于有损压缩格式。MPEG1 除采用了 JPEG 的所有技术以外，还引进了自适应量化、运动补偿预测、双向运动补偿及半像素运动估计等技术，实现了对动态图像的高效压缩。

MPEG2 是一个直接与数字电视广播有关的高质量图像和声音编码标准。可以说是 MPEG1 的扩展，二者的基本编码算法相同，但是 MPEG2 增加了许多 MPEG1 所没有的功能，如隔行扫



描电视编码、位速率可变性能等。MPEG2 的传输速率达 4～15Mbps。

MPEG4 是为视听（audio—visual）数据的编码和交互播放而开发的算法和工具，它是一个数据速率很低（4.8～32Kbps）的多媒体通信标准。MPEG4 在异构网络环境下能够高度可靠地工作，并且具有很强的交互功能。为此，MPEG4 引入了对象基表达(object—based represe— ntation)的概念，用来表达视听对象(audio/visual objects，AVO)。MPEG4 扩充了编码的数据类型，由自然数据对象扩展到计算机生成的合成数据对象，采用合成对象/自然对象混合编码(synthetic/natural hybrid coding，SNHC)算法。在实现交互功能和重用对象中引入了组合、合成和编排等重要概念。MPEG7 与 MPEG1、MPEG2、MPEG4 不同，它是多媒体内容描述接口(Multimedia Content Description Interface)，致力于制定一个标准化的框架，用来描述各种类型的多媒体信息及它们之间的关系，以便更快更有效地检索信息。这些媒体材料可包括静态图像、图形、3D 模型、声音、话音、电视，以及在多媒体演示中它们之间的组合关系。MPEG7 标准是建立在其他标准之上的，例如，PCM，MPEG1，MPEG2 和 MPEG4 等等。在 MPEG7 中，用到了 MPEG4 中使用的形状描述符、MPEG1 和 MPEG2 中使用的运动向量（motion vector）等。MPEG21 是一个多媒体框架（multimedia framework），允许用户可以透明地定制使用各种类型网络上的多媒体资源。MPEG21 提出的数据项目声明（digital item declaration）作为网上媒体资源打包成数据项目的抽象，它适用于各种不同的终端和网络资源。目前的 MPEG4 和 MPEG7 框架都可以集成到 MPEG21 框架中。关于 JPEG、MPEG 等的编码压缩方式的具体内容，请参阅其他书籍。

List 12.1 预测编码（DPCM）

```
#include "StdAfx.h"
#include "BaseList.h"

#define B_VAL 128     //图像区域以外时的像素值

/*--- Dpcm1 --- 预测编码 DPCM （预测法(1) 处理一行区域）--------
   image_in:  输入图像数据指针
   xsize:     输入图像宽度
   line:      输入行序号
   data_out:  输出一行的 DPCM 数据指针
--------------------------------------------------------------*/
void Dpcm1(BYTE* image_in, int xsize, int line, short *data_out)
   {
   int pred ,i;

   for (i = 0; i < xsize; i++) {
```



```
    if (i == 0) pred = B_VAL;
    else       pred = (int)(*(image_in + line*xsize + i-1));
    *(data_out + i) = (int)(*(image_in + line*xsize + i)) - pred;
  }
}

/*--- Dpcm2 --- 预测编码 DPCM (预测法(2) 处理一行区域) --------
  image_in: 输入图像数据指针
  xsize:    输入图像宽度
  line:     输入行序号
  data_out: 输出横线的 DPCM 数据指针
---------------------------------------------------------------*/
void Dpcm2(BYTE* image_in, int xsize, int line, short *data_out)
{
  int pred, i;

  if (line == 0) {          //第一行时
    for (i = 0; i < xsize; i++) {
      if (i == 0) pred = B_VAL;
      else       pred = (*(image_in + line*xsize + i-1) + B_VAL) / 2;
      *(data_out + i) = (int)(*(image_in + line*xsize + i)) - pred;
    }
  }
  else {                    //其他行
    for (i = 0; i < xsize; i++) {
      if(i==0)pred=(B_VAL + *(image_in + (line-1)*xsize +i)) / 2;
      else pred=(*(image_in+line*xsize+i-1)+*(image_in+ (line-1)*xsize + i)) / 2;
      *(data_out + i) = *(image_in + line*xsize + i) - pred;
    }
  }
}
```



List 12.2　DPCM数据直方图

```
#include "StdAfx.h"
#include "BaseList.h"

#define DPCM   Dpcm1   // 预测法(1), 预测法(2)时替换为dpcm2

/*--- Histgram_dpcm --- DPCM 数据分布直方图 -----------------------------
   image:     图像数据指针
   xsize:     图像宽度
   ysize:     图像高度
   hist:      直方图配列
-------------------------------------------------------------------*/
   void Histgram_dpcm(BYTE *image, int xsize, int ysize, long hist[512])
   {
   int i,j,n;
   short *data_out;

   data_out = new short[xsize];

   for (n = 0; n < 512; n++) hist[n] = 0;
   for (j = 0; j < ysize; j++) {
     DPCM(image, xsize, j, data_out);

   for ( i = 0; i < xsize; i++) {
       n = data_out[i] + 255;
       hist[n]++;
     }
   }

   delete [] data_out;
}
```

List 12.3　DPCM数据直方图的百分比分布

```
#include "StdAfx.h"
#include "BaseList.h"
```



```
/*--- CalHistPercent_dpcm --- 计算 DPCM 直方图百分比 ----------------------
  hist:           输入 DPCM 直方图数列
  hist_radio:     输出百分比 DPCM 直方图数列
  max_percent:    输出 DPCM 直方图最大百分比

-----------------------------------------------------------------------*/

void CalHistPercent_dpcm(long hist[],float hist_radio[],float &max_percent)
{
   float max_value;
   short i;
   float total = (float)0;

   for( i=0 ; i<512 ; i++ )
     total = total + (float)hist[i];

   max_value = 0;//初始化
   for( i=0 ; i<512 ; i++ ){
     //计算比例
   if(total > 0)
   hist_radio[i] =  ((hist[i]/total)*(float)100);
   //求最大像素数
   if( max_value < hist[i] )
      max_value = (float)hist[i];
 }

 //求最大比例值
 max_percent = ((max_value/total)*(float)100);

}
```

List 12.4 解码程序

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Idpcm1 --- DPCM 的解码 (预测法(1), 处理一行区域) -------------
  data_in:     输入的一行 DPCM 符号数据指针
```



```
   xsize:        输入图像宽度
   line:         输入行序号
   image_out:    输出图像数据指针
-------------------------------------------------------------------*/
void Idpcm1(short *data_in, int xsize, int line, BYTE* image_out)
{
   int pred;             //预测值
   int i;

   for (i = 0; i < xsize; i++) {
     if (i == 0) pred = B_VAL;
     else        pred = (int)(*(image_out + line*xsize + i-1));
     *(image_out + line*xsize + i) = (BYTE)(pred + (int)(*(data_in + i)));
   }
}

/*--- Idpcm2 --- DPCM的解码(预测法(2)，处理一行区域)-------------
   data_in:    输入的一行DPCM符号数据指针
   xsize:      输入图像宽度
   line:       输入行序号
   image_out:  输出图像数据指针
-------------------------------------------------------------------*/
void Idpcm2(short *data_in, int xsize, int line, BYTE* image_out)
{
   int pred, i;

   if (line == 0) {              //第一行
     for (i = 0; i < xsize; i++) {
       if (i == 0) pred = B_VAL;
       else        pred=((*(image_out + line*xsize+i-1))+B_VAL) / 2;
       *(image_out + line*xsize + i) = pred + *(data_in + i);
     }
   }
   else {                  //其他行
     for (i = 0; i < xsize; i++) {
      if(i == 0)pred = (B_VAL + *(image_out + (line-1)*xsize + i)) / 2;
      else pred=(*(image_out+line*xsize+i-1)+*(image_out+ (line-1)*xsize + i)) / 2;
```



```
        *(image_out + line*xsize + i) = pred + *(data_in + i);
      }
    }
}
```

List 12.5　变长编码

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Vlcode --- 变长编码 ---------------------------------------------
   data_in:    输入数据
   no:         数据
   vlc_out:    输出可变长符号
------------------------------------------------------------------------*/
int Vlcode(short int data_in[], int no, char vlc_out[])
{
   int  i;
   int  st = 0;
   int  num = 0;                    //符号长（字节）
   int  dl = BYTESIZE / LEN - 1;
   int  mask = (1 << LEN) - 1 ;
   int  dt, ms;

   vlc_out[num] = '\0';
   for(i = 0; i < no; i++) {
     dt = data_in[i];
     do {
       ms = dt >= mask ? mask : dt;
       vlc_out[num] |= (ms << (LEN * (dl - st)));
       dt -= mask;  st++;
       if(st > dl) {
         st = 0;  num++; vlc_out[num] = '\0';
       }
     } while(dt >= 0);
   }
   if(st != 0) {        //将字节最后剩余位置1
     ms = mask;
```



```
      for(i = (dl - st); i >= 0; i--) {
        vlc_out[num] |= ms;
        ms <<= LEN;
      }
      num++;
    }
    return num;
  }
```

List 12.6　变长编码的解码

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Ivlcode --- 变长编码的解码 --------------------------------------
   vlc_in:     输入可变长符号
   no:         输入数据
   data_out:   输出数据
------------------------------------------------------------------*/
void Ivlcode(char vlc_in[], int no, short int data_out[])
{
   int  i, j, k;
   int  ino = 0;          //符号长（字节）
   int  num = 0;
   int  dl = BYTESIZE / LEN - 1;
   int  mask = (1 << LEN) - 1;
   for(i = 0; i < no; i++) data_out[i] = 0;
     do {
       for(j = dl; j >= 0; j--) {
       k = vlc_in[ino] & (mask << (LEN * j));
       k >>= (LEN * j);
       data_out[num] += k;
       if(k != mask) num++;
       if (num >= no) break;
     }
     ino++;
   } while (num < no);
}
```



List 12.7 变长码与DPCM码之间的转换

```
#include "StdAfx.h"
#include "BaseList.h"

/*--- Event --- 由 DPCM 码到变长码的变换 -------------------------
 dt:    输入 DPCM 数据
-------------------------------------------------------------------*/
int Event(short dt)
{
   int  ev;

   if(dt <= 0) ev = -2 * dt;
   else        ev = 2 * dt - 1;
   return ev;
}

/*--- Ievent ---由变长码到 DPCM 码的转换-----------------------------
   ev:    输入可变长符号值
-------------------------------------------------------------------*/
int Ievent(short ev)
{
   int  dt;

   if(ev % 2 == 0) dt = -ev / 2;
   else            dt = (ev+1) / 2;
   return dt;
}
```

List 12.8 DPCM + 变长编码

```
#include "StdAfx.h"
#include "BaseList.h"

#define DPCM   Dpcm1   // 预测法(1), 预测法(2)时替换为 dpcm2

/*--- Dpcm_vlcode --- DPCM + 变长编码 -------------------------------
   image_in:    输入图像数据指针
   xsize:       图像宽度
```



```
   ysize:        图像高度
   image_buf:  输出图像数据指针（符号化列）
------------------------------------------------------------------*/
int Dpcm_vlcode(BYTE *image_in, int xsize, int ysize, BYTE* image_buf)
{
   int      i, j, leng;
   long     ptr, size;
   long        max_leng;  //变长码用配列的最大字节数
   char     *vlc;     //变长码

   short *data;      //一行的 DPCM 数据

   data = new short [xsize];

   size = (long)xsize*ysize;
   max_leng = (long)xsize*4;

   vlc = new char[max_leng];

   for (i = 0; i < size; i++) image_buf[i] = 0;
   ptr = 0;
   for (j = 0; j < ysize; j++) {        //一行一行编码
     DPCM(image_in, xsize, j  data);   //预测编码
     for(i = 0; i < xsize; i++)
       data[i] = Event(data[i]);      //转换为变长码
     leng = Vlcode(data, xsize, vlc);    //变长编码
     image_buf[ptr] = (unsigned char)((leng >> 8) & 0x00ff);
     ptr++;     //如果码字数据比原图像大，数据错误返回-1
     if (ptr > size)
     {
       delete [] data;
       delete [] vlc;
       return -1;
     }
     image_buf[ptr] = (unsigned char)(leng & 0x00ff);
     ptr++;     //如果码字数据比原图像大，数据错误返回-1
     if (ptr > size)
```



```
    {
      delete [] data;
      delete [] vlc;
      return -1;
    }
    for (i = 0; i < leng; i++) {
      image_buf[ptr] = vlc[i];
      ptr++;    //如果码字数据比原图像大，数据错误返回-1
      if (ptr > size)
      {
        delete [] data;
        delete [] vlc;
        return -1;
      }
    }
  }

  delete [] data;
  delete [] vlc;
  return ptr;
}
```

List 12.9 DPCM + 变长编码的解码

```
#include "StdAfx.h"
#include "BaseList.h"
#include <math.h>
#include <stdio.h>

#define IDPCM    Idpcm1    // 预测法(1)，预测法(2)时替换为 Idpcm2

/*--- Idpcm_vlcode --- DPCM + 变长编码的解码-------------------------------
  image_buf:    输入图像数据（码字数据）
  image_out:    输出图像数据
  xsize:        输入图像宽度
  ysize:        输入图像高度
------------------------------------------------------------------------*/
int Idpcm_vlcode(BYTE *image_buf, BYTE *image_out, int xsize, int ysize)
```



```
{
  int      i, j, leng;
  long     ptr, size;
    long       max_leng;  //变长编码用配列的最大字节数
  char     *vlc;     //变长码字数据

  short *data;      //一行的 DPCM 数据

  data = new short[xsize];

  size = (long)xsize*ysize;
  max_leng = (long)xsize*4;

  vlc = new char[max_leng];

  ptr = 0;
  for(j = 0; j < ysize; j++) {    //一行一行解码
    leng = (int)image_buf[ptr];
    ptr++;    //如果码字数据比原图像大，数据错误返回-1
    if (ptr > size) return -1;
    leng = (leng << 8) | image_buf[ptr];
    ptr ++;
    for (i = 0; i < leng; i++) {
      vlc[i] = image_buf[ptr];
      ptr++;
      if (ptr > size) return -1;
    }     //如果码字数据比原图像大，数据错误返回-1
    Ivlcode(vlc, xsize, data);      //变长码解码
    for(i = 0; i < xsize; i++)
      data[i] = Ievent(data[i]);    //返变长码字为差分数据
    IDPCM(data, xsize, j, image_out); //由差分数据复原图像数据
  }

  delete [] data;
  delete [] vlc;
  return 0;
}
```





[2] 根据设计需要，把元素 element1 改成有意义的名称：Products。

[3] 单击元素第一行的第一列，展开下拉列表，如图 10-7 所示。

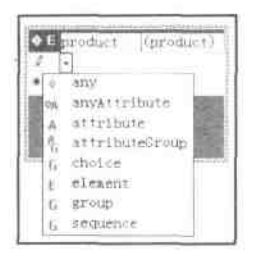


图 10-7

[4] 在下拉列表中选择元素。XML 设计器把一个嵌套元素添加到 Products 元素中，将该元素名称改为 Productid。

[5] 用同样的方法再在表中加一个嵌套元素 Categoryid。

2 / 把表或视图添加到架构中


操作步骤

[1] 在 XML 设计器中打开 Dataset1 架构，然后选择"数据集"选项卡。

[2] 在"服务器资源管理器"中，展开与 SQL Northwind 数据库的连接，然后展开"表"节点。

[3] 选中 Categories 表并拖到 XML 设计器中，VB.NET 会在架构中添加一个表，如图 10-8 所示。

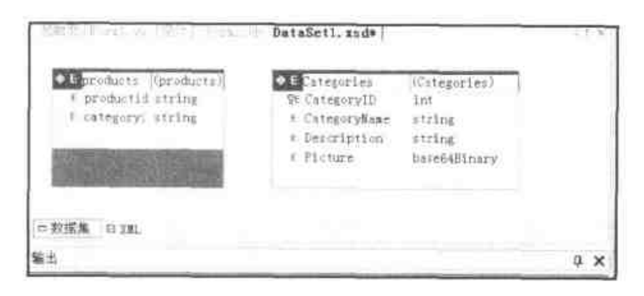


图 10-8

3. 创建键

XML 设计器中有 3 个不同的标记适用于实体和引用的完整性：主键（primary key）、键引用（keyref）和惟一键（unique key）。主键保证了 Dataset 中的惟一性。Keyref 标记代表一个外键的引用，用来完成一个一对多的关系。惟一键保证惟一性，但是一般



到目前为止，一般的信号分析与合成中，经常使用第 11 章所介绍的傅立叶变换（Fourier transform）。然而，由于傅立叶变换的基（basis）是采用无限连续且不具有局部性质的三角函数，所以在经过傅立叶变换后的频率域中时间信息完全丢失。与其相对，本章将要介绍的小波变换，由于其能够得到局部性的频率信息，从而可以有效地进行时间频率分析。

小波据说是法国人 Morlet 在以石油勘探为目的的人工地震波分析中导入的，而后很多数学家和工程技术人员都对其进行研究，在图像处理领域也得到了广泛的应用。

13.1　连续小波变换

在实数 R 上定义函数 $f(t)$ 时，把满足以下条件的函数称为平方可积函数（square integrable function）。

$$\int_{-\infty}^{\infty}\left|f(t)\right|^2 \mathrm{d}t < \infty \tag{13.1}$$

即 $f(t)$ 需要在无限远处收缩。平方可积函数的空间用 $L^2(R)$ 表示。这意味着 11.2 节的傅立叶变换的积分核 $e^{-j\omega x}=\cos\omega x-j\sin\omega x$ 不属于 $L^2(R)$，其中 ω 为角频率。傅立叶级数展开是利用 $e^{jn\omega_0 t}$ 的线性组合来表示（π，−π）区间上的任意信号的。而小波变换是以属于 $L^2(R)$ 的函数 $\psi(t)$ 作为基（basis）来表现属于 $L^2(R)$ 的任意信号的。

小波（wavelet）意思是小的波或者细的波。如图 13−1 所示，$\psi(t)$ 被定义为具有局部性质的波，且作为波的基本单位使用，被称为基本小波（basic wavelet）或者母小波（mother wavelet），或者简称为小波（wavelet）。小波函数（wavelet function）是利用两个实数参数 a 和 b 以下式来定义的

$$\psi_{a,b}(t)=\frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{13.2}$$

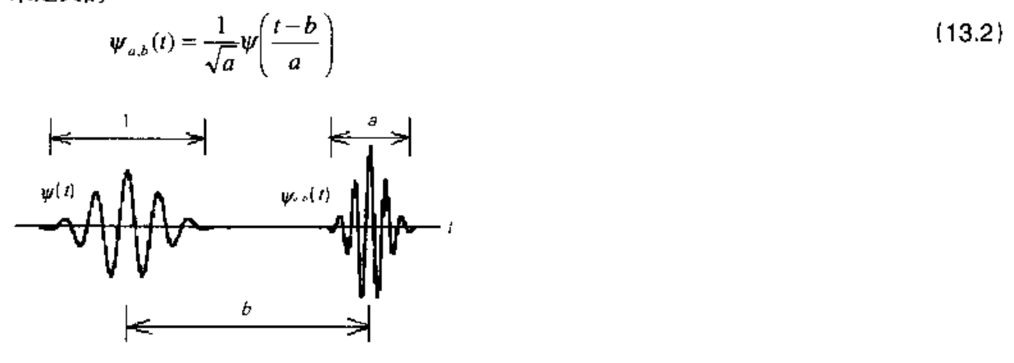


图 13-1

如图 13−1 所示，小波函数是小波 $\psi(t)$ 在水平方向增加到 a 倍，平移 b 的距离得到的。$1/\sqrt{a}$



是用于规范化（normalization，归一化）的系数、a为尺度参数（scale），b为平移参数（shift）。由于a表示小波的时间幅值，所以$1/a$相当于频率。

水平坐标t作为变量，如下所示的任意信号$f(t)$与小波函数$\psi_{a,b}(t)$的内积（inner product，点积）被称为小波变换（wavelet transform）或者连续小波变换（continuous wavelet transform，CWT）。

$$W(a,b)=\left\langle f(t),\psi_{a,b}(t)\right\rangle=\frac{1}{\sqrt{a}}\int_R f(t)\psi^*\left(\frac{t-b}{a}\right)\mathrm{d}t \tag{13.3}$$

小波函数一般是复数（complex number），其内积中使用复共轭（a pair of complex number）。$W(a,b)$相当于傅立叶变换的傅立叶系数，$\psi^*(\cdot)$为$\psi(\cdot)$的复共轭，$t-b$时表示信号$f(t)$中包含有多少$\psi_{a,b}(t)$的成分。a值小时对应高频成分，a值大时对应低频成分。也就是说，对于高频成分，频率分辨率小，时间（位置）分辨率大；对于低频成分，频率分辨率增大，可是时间（位置）分辨率却减小。

下面考虑其逆变换。首先C_ψ如下来定义。

$$C_\psi=\int_{-\infty}^{\infty}\frac{\left|\hat{\psi}(\omega)\right|^2}{|\omega|}\mathrm{d}\omega \tag{13.4}$$

在此$\hat{\psi}(\omega)$是$\psi(t)$的傅立叶变换，如下式所示。

$$\hat{\psi}(\omega)=\int_{-\infty}^{\infty}\psi(t)e^{-j\omega t}\mathrm{d}t \tag{13.5}$$

为了确保逆连续小波变换（inverse continuous wavelet transform）的存在，必须满足下面的条件：

$$\int_0^{\infty}\frac{\left|\hat{\psi}(\omega)\right|^2}{\omega}\mathrm{d}\omega\ \int_{-\infty}^{0}\frac{\left|\hat{\psi}(\omega)\right|^2}{\omega}\mathrm{d}\omega=\frac{1}{2}C_\psi<\infty \tag{13.6}$$

这时由于$\hat{\psi}(\omega)$有界连续，所以为了使$C_\psi<\infty$成立，需要$\hat{\psi}(0)=0$。也就是$\psi(t)$的直流成分必须为0，即平均值必须为0。

这个条件成立的话，下面的逆变换成立。

$$f(t)=\frac{2}{C_\psi}\int_0^{\infty}\left[\int_{-\infty}^{\infty}W(a,b)\psi_{a,b}(t)\mathrm{d}b\right]\frac{\mathrm{d}a}{a^2} \tag{13.7}$$

也就是说，通过基$\psi_{a,b}(t)$就能够表现信号$f(t)$。然而，这个表现信号在重构时需要基于a、b的无限积分，这是不切实际的。在进行基于数值计算的信号小波变换以及逆变换时，需要使用离散小波变换（discrete wavelet transform，DWT）。

从式（13.3）可见，将取正弦波和余弦波的合成$e^{j\omega t}$作为积分核，再取内积的傅立叶级数，变成取（基本）小波作为基，再取内积的变换即为小波变换。换句话说，在傅立叶级数中，是检测所给函数$f(t)$相似于正弦波或者余弦波的程度。然而，由于正弦波或者余弦波是无限连续的函数，所以如果所给函数$f(t)$在时间轴上是局部性函数的话，就可能与正弦波或者余弦波



那样的无限连续函数没有任何相似性。

因此，如果采用在时间轴上相似于局部性函数 $f(t)$ 的局部性的波，即小波，就可能适当地检测出时间轴上的局部性波形。这就是小波的出发点。

13.2 二阶小波

ψ 为基本小波（即小波）时，对于函数 $f(t)$ 和实数 $a,b\in R$（$a>0$），把下面的积分称为小波变换：

$$W(a,b)=\frac{1}{\sqrt{a}}\int R\ f(t)\psi^*\left(\frac{t-b}{a}\right)dt \tag{13.8}$$

上式曾经在13.1中介绍过，即式（13.3）。在此把参数因子 a、b 按下式取二进分割（binary partition），即可对连续小波离散化

$$\begin{aligned} a&=2^j \\ b&=k2^j \end{aligned} \tag{13.9}$$

如 j=0，±1，±2，…离散化时，相当于小波函数的宽度减少一半，进一步减少一半，或者增加一倍 进一步增加一倍等进行伸缩。另外，由 k=0，±1，±2，…能够覆盖所有的变量领域。

把式（13.9）代入式（13.2）得到的小波函数称为二阶小波（dyadic wavelet），即：

$$\psi_{j,k}(t)=\frac{1}{\sqrt{2^j}}\psi\left(\frac{t-k2^j}{2^j}\right)=2^{-\frac{j}{2}}\psi(2^{-j}t-k) \tag{13.10}$$

采用这个公式的小波变换称为离散小波变换（discrete wavelet transform）。这个公式是Daubechies的表示法，t 前面的 2^{-j} 相当于傅立叶变换的角频率，所以 j 值较小的时候为高频。另一方面，在Meyer表示法中，j 的前面没有负号，所以与Daubechies表示法相反，j 值越大则频率越高。这个 j 被称为级（level）或分辨率索引。这两种表现法的存在容易造成混乱，所以在后续的13.4节将介绍各自用法。

适当选取式（13.10）的 ψ 就可以使 $\{\psi_{j,k}\}$ 成为正交系。首先有关平移（translation）的正交性，对 k 进行整数平移的 $\psi(t-k)$ 和对 n 进行整数平移的 $\psi(t-n)$ 满足如下的关系：

$$\begin{aligned}\langle\psi(t-k),\psi(t-n)\rangle&=\int_{-\infty}^{\infty}\psi(t-k)\psi^*(t-n)dt\\ &=\begin{cases}c & (n=k)\\ 0 & (n\neq k)\end{cases}\end{aligned} \tag{13.11}$$

其中：

$$\langle\psi(t),\psi(t)\rangle=\int_{-\infty}^{\infty}|\psi(t)|^2 dt=c \tag{13.12}$$

也就是说，正交条件是指仅平移某个整数 n 的函数和仅平移 k 整数的函数的内积，平移量



$n=k$ 以外的情况都为 0。另外，上式中使 $c=1$ 规范化（归一化）的情况被特别称为规范正交（orthonormal）。

关于放大缩小（dilation）的规范正交是指二进放大缩小 $2^{-\frac{m}{2}}\psi(2^{-m}t)$ 和 $2^{-\frac{n}{2}}\psi(2^{-n}t)$，应满足下式：

$$\left\langle 2^{-\frac{m}{2}}\psi(2^{-m}t), 2^{-\frac{n}{2}}\psi(2^{-n}t)\right\rangle=\begin{cases}1 & (m=n)\\ 0 & (m\neq n)\end{cases} \tag{13.13}$$

同时能满足上述平移和放大缩小正交条件的，最简单的小波是哈尔小波（Haar wavelet），用公式表示为式（13.14），用图表示为图 13-2。

$$\psi(t)=\begin{cases}1 & (0\leqslant t<1/2)\\ -1 & (1/2\leqslant t<1)\\ 0 & (\text{其他情况})\end{cases} \tag{13.14}$$

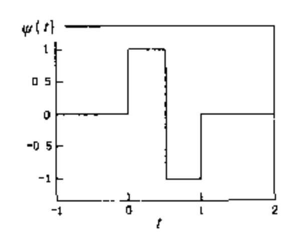


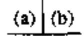
图 13-2

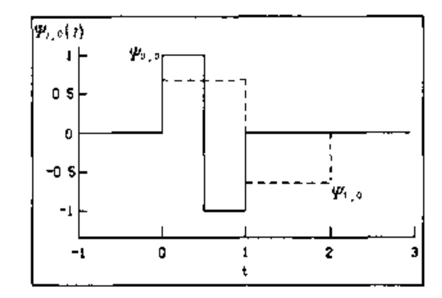
用图 13-3（a）说明了哈尔小波满足放大缩小的规范正交条件。由图中的 $\psi_{0,0}$ 和 $\psi_{1,0}$ 的内积得到：

$$\langle\psi_{0,0},\psi_{1,0}\rangle=1\times1/\sqrt{2}\times0.5-1\times1/\sqrt{2}\times0.5-0\times1/\sqrt{2}\times1=0 \tag{13.15}$$

由 $\psi_{0,0}$ 和 $\psi_{0,0}$ 的内积，可知：

$$\langle\psi_{0,0},\psi_{0,0}\rangle=1^2\times0.5+(-1)^2\times0.5=1 \tag{13.16}$$





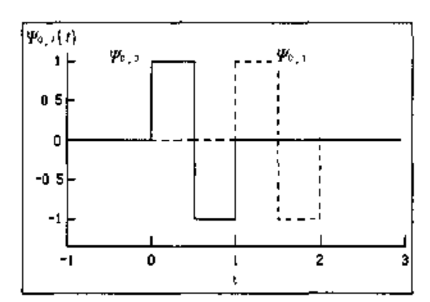


图 13-3

另外，平移也一样，如图 13-3（b）所示，对 $\psi_{0,0}$ 进行 $k=1$ 平移，即得到 $\psi_{0,1}$。$\psi_{0,0}$ 和 $\psi_{0,1}$ 之间没有任何重合之处，故满足规范正交条件。进而，对放大缩小和平移组合在一起的情况，由图 13-3（a）可知，$\psi_{0,0}$ 无论是在正负哪个方向上的整数平移，$\psi_{0,k}$ 也只是在 $\psi_{1,0}$ 的平坦范围内振动。因此，$\psi_{1,0}$ 和 $\psi_{0,k}$ 的内积为 0，满足正交条件。

满足这个正交条件时信号 $f(t)$ 可展开为以小波为基的级数，如下所示：

$$f(t)=\sum_{j}\sum_{k}w_k^{(j)}\psi_{j,k}(t) \tag{13.17}$$

如果这个小波系数 $w_k^{(j)}$ 是规范正交的话，它将以内积的形式给出：

$$w_k^{(j)}=\int_{-\infty}^{\infty}f(t)\psi_{j,k}^{*}(t)\mathrm{d}t=\left\langle f,\psi_{j,k}\right\rangle \tag{13.18}$$

这个积分的含义是：在傅立叶级数的系数中，取正弦或者余弦函数的内积，来求正弦或者余弦函数和信号的相关性。与之相对，在此代替正弦或者余弦函数使用离散小波，检测小波和信号的相关性。

本章中使用 Daubechies 表示法，同样小波系数（wavelet coefficient）像 $w_k^{(j)}$ 那样平移 k 被表示在右下方，尺度 j 加括号表示在右上方。

13.3 信号的分解与重构

下面使用小波系数（wavelet coefficient）说明信号分解与重构（decomposition and reconstruction）的方法。

首先，由被称为尺度函数的线性组合来近似表示信号。尺度函数的线性组合称为近似函数（approximate function）。另外，近似的精度被称为级（level）或分辨率索引，第 0 级是精度最高的近似，级数越大表示粗略程度越高。任意第 j 级的近似函数与第 $j+1$ 级的近似函数之间的差分就是小波的线性组合。信号最终可以由第 1 级开始到任意级的小波与尺度函数的线性组合来表示。

宽度为 1 的矩形脉冲作为尺度函数 $\varphi(t)$，由这个函数的线性组合生成任意信号 $f(t)$ 的近似函数 $f_0(t)$，如下所示：

$$f_0(t)=\sum_{k}s_k\varphi(t-k) \tag{13.19}$$

其中：

$$\varphi(t)=\begin{cases}1 & (0\leqslant t<1)\\ 0 & (\text{其他情况})\end{cases} \tag{13.20}$$

系数 s_k 是区间 $[k,k+1]$ 内信号 $f(t)$ 的平均值，由下式给出



$$s_k=\int_{-\infty}^{\infty}f(t)\varphi^*(t-k)\mathrm{d}t=\int_k^{k+1}f(t)\mathrm{d}t \tag{13.21}$$

信号 $f(t)$ 的例子及其近似函数 f_0 显示在图 13–4 中。如图 13–5 所示，为生成近似函数所用的宽度为 1 的矩形脉冲 $\varphi(t)$，由于其功能是作为观测信号的尺度，所以被称为尺度函数（scaling function）。在此被特别称为哈尔尺度函数（Haar's scaling function）。

与小波相同，考虑尺度函数的整数平移及放大缩小，$\varphi_{j,k}$ 如下式定义：

$$\varphi_{j,k}(t)=2^{-\frac{j}{2}}\varphi(2^{-j}t-k) \tag{13.22}$$

哈尔尺度函数的整数平移也是规范正交，用下式表示。

$$\begin{aligned}\langle\varphi(t-k),\varphi(t-n)\rangle&=\int_{-\infty}^{\infty}\varphi(t-k)\varphi^*(t-n)\mathrm{d}t\\&=\begin{cases}1 & (k=n)\\0 & (k\neq n)\end{cases}\end{aligned} \tag{13.23}$$

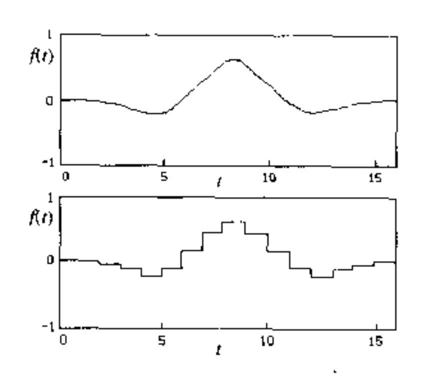


图 13-4

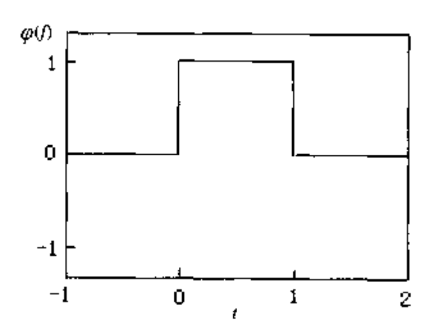


图 13-5

如图 13–6 所示，表示了 k=0、n=2 的例子。由此可见，$\varphi(t)$ 和 $\varphi(t-n)$ 之间没有重合之处，式（13.23）的成立在视觉上是容易理解的。

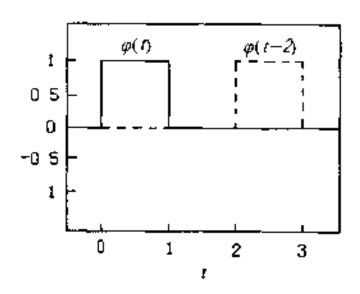


图 13-6



下面，使用$\varphi_{j,k}$定义第j级的近似函数$f_j(t)$，如下式所示：

$$f_j(t)=\sum_k s_k^{(j)}\varphi_{j,k}(t) \tag{13.24}$$

其中：

$$s_k^{(j)}=\int_{-\infty}^{\infty} f(t)\varphi_{j,k}^*(t)\mathrm{d}t \tag{13.25}$$

另外，由于$\varphi_{j,k}(t)$对于平移是规范正交的，所以$s_k^{(j)}$是由第j级的近似函数f_j和尺度函数$\varphi_{j,k}$的内积求得的，用下式表示：

$$s_k^{(j)}=\int_{-\infty}^{\infty} f_j(t)\varphi_{j,k}^*(t)\mathrm{d}t \tag{13.26}$$

这个$s_k^{(j)}$被称为尺度系数（scaling coefficient）。图13-7表示了信号$f(t)$及其近似函数$f_0(t)$和$f_1(t)$。

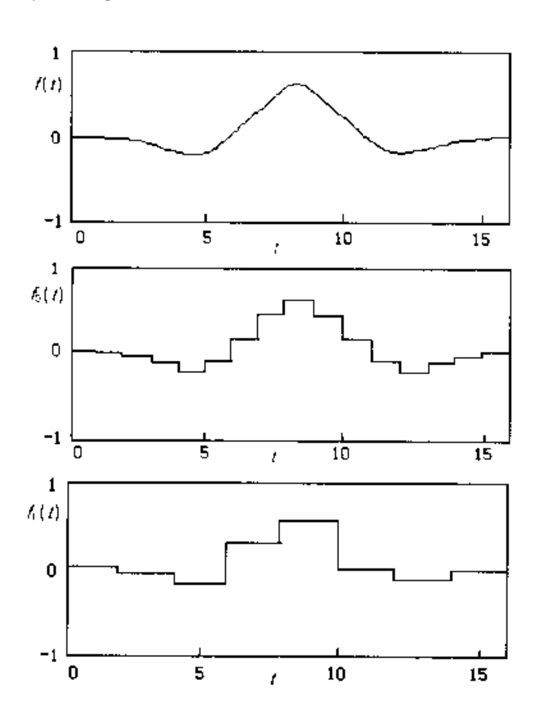


图 13-7

比较$f_0(t)$和$f_1(t)$，很明显$f_1(t)$的信号是更加粗略的近似。$f_1(t)$在式（13.22）中是$j=1$的情况，t前面的系数为2^{-1}，该系数是$j=0$时的一半。这个系数相当于傅立叶变换的角频率，所以尺度函数φ的宽度成为$j=0$时的2倍。因此，$f_1(t)$的情况是想用更宽的矩形信号来近似表示



信号 $f(t)$，这时由于无法表示细致的信息，造成了信号分辨率下降。

由于用 f_1 近似表示（或逼近）f_0 时有信息脱落，所以只有用被脱落的信息 $g_1(t)$ 来弥补 $f_1(t)$，才能够使 $f_0(t)$ 复原。即：

$$f_0(t)=f_1(t)+g_1(t) \tag{13.27}$$

$g_1(t)$ 是从图 13–7 的 f_0 减去 f_1 所得的差值，如图 13–8 所示。

函数 $g_1(t)$ 被称为第 1 级的小波成分（wavelet component）。

由图 13–8 可知，左右宽度为 1 的区间是正负对称而上下振动的，因此 $g_1(t)$ 的构成要素一定是以下所示的函数

$$\psi\left(\frac{t}{2}\right)=\begin{cases}1 & (0\leqslant t<1)\\ 1 & (1\leqslant t<2)\\ 0 & (\text{其他情况})\end{cases} \tag{13.28}$$

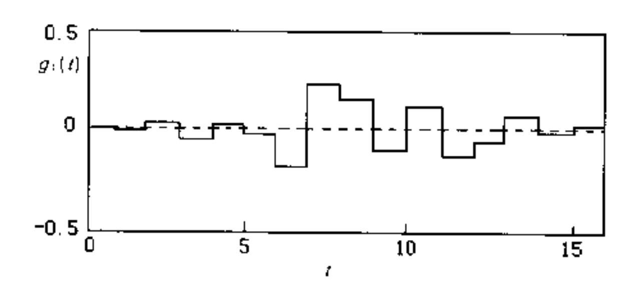


图 13-8

可见，这个 $\psi(t)$ 只能是上节中式（13.14）所示的哈尔小波。这个哈尔小波可按照上节所述的那样通过式（13.10）的放大缩小和平移来生成函数族 $\psi_{j,k}$。

在第 1 级（j=1）时，由 $\psi_{j,k}$ 的线性组合按下式表示 $g_1(t)$：

$$g_1(t)=\sum_k w_k^{(1)}\psi_{1,k}(t) \tag{13.29}$$

其中 $w_k^{(1)}$ 是第 1 级（j=1）的小波系数。

综上所述，第 0 级的近似函数可以分解为第 1 级的尺度函数的线性组合 $f_1(t)$ 与第 1 级小波的线性组合 $g_1(t)$，如下式所示：

$$\begin{aligned}f_0(t)&=f(t)+g_1(t)\\&=\sum_k s_k^{(1)}\varphi_{1,k}(t)+\sum_k w_k^{(1)}\psi_{1,k}(t)\end{aligned} \tag{13.30}$$

把这个关系扩展到第 j 级的一般情况。即从第 j 级的近似函数 f_j 来生成精度高一级的第 j–1



级的近似函数 f_{j-1} 时，只需求第 j 级的近似函数 $f_j(t)$ 与小波成分 $g_j(t)$ 的和即可：

$$f_{j-1}(t) = f_j(t) + g_j(t) \tag{13.31}$$

其中

$$\begin{aligned} f_j(t) &= \sum_k s_k^{(j)} \varphi_{j,k}(t) \\ g_j(t) &= \sum_k w_k^{(j)} \Psi_{j,k}(t) \end{aligned} \tag{13.32}$$

下面考虑把第 0 级的近似函数 $f_0(t)$ 用精度一直降到第 J 级的近似函数来表示。在式(13.31)中代入 j=1，2，…，J 得

$$\begin{aligned} f_0(t) &= f_1(t) + g_1(t) \\ f_1(t) &= f_2(t) + g_2(t) \\ &\cdots \\ f_{J-1}(t) &= f_J(t) + g_J(t) \end{aligned} \tag{13.33}$$

在上式中把最下面的 $f_{J-1}(t)$ 代入邻接的上式中所得到的式子，再代入其邻接的上式中，不断重复迭代上述操作直到 $f_0(t)$ 为止，可见 $f_0(t)$ 可以用 $f_J(t)$ 和 $g_j(t)$ 集合的和表示，如下式所示

$$\begin{aligned} f_0(t) &= g_1(t) + g_2(t) + \cdots + g_J(t) + f_J(t) \\ &= \sum_{j=1}^{J} g_j(t) + f_J(t) \end{aligned} \tag{13.34}$$

这个公式的含义是：把信号 $f_0(t)$ 用第 J 级的近似函数 $f_J(t)$ 来粗略近似地表示时，如果把粗略近似所失去的成分顺次附加上去的话，就可以恢复 $f_0(t)$。也就是说，信号 $f_0(t)$ 能够表现为任意粗略级的近似函数 $f_J(t)$ 和第 0 级到第 J 级的小波成分的和。因此可以说，信号 $f_0(t)$ 能够用从第 1 级到第 J 级的 J 个分辨率即多分辨率的小波来表示。这种信号分析称为多分辨率分析（multi—resolution analysis）。

图 13—9 表示了 J=2 时的多分辨率分析的例子。在这个例子中，$f_0=g_1+g_2+f_2$ 的关系成立。这样，f_2 是呈矩形的形状，可是如果加大 J 的话，矩形的宽度还将拉伸得比 f_2 更宽。因此，信号中含有直流成分（平均值非 0）时，有必要把这个直流成分用 f_J 来表示，与直流重合的振动部分用小波来表示。因为平均值为 0 的小波的线性组合，平均值还是 0，所以用有限个小波是无法表示直流成分的。



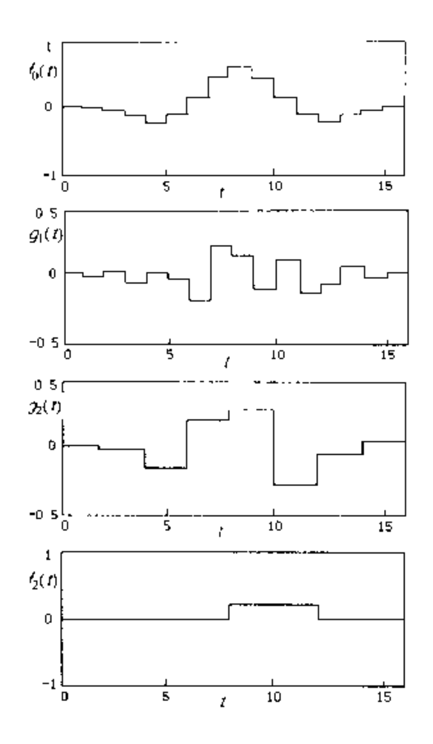


图 13-9

13.4 双尺度关系

哈尔小波的例子表明：只要确定了尺度函数，依公式（13.27）就可以导出小波，即 $g_1(t)=f_0(t)-f_1(t)$。那么，可以把这个关系扩展到哈尔小波以外的小波。但是不是任何函数都可得到尺度函数，再从这个尺度函数导出小波来呢？根据多分辨率解析的定义，答案是否定的。构成多分辨率分析的必要条件是：第 j 级的尺度函数 $\varphi_{j,k}$ 能够用精度高一级的第 $j-1$ 级的尺度函数 $\varphi_{j-1,k}$ 来展开。如果用数学式 2 表示，平移 $k=0$ 时如下式所示

$$\varphi_{j,0}(t)=\sum_{n}P_n\varphi_{j-1,n}(t) \tag{13.35}$$

其中，序列 p_n 为展开系数。这个式子表示：第 j 级的尺度函数可以用第 $j-1$ 级即精度高一级的尺度函数表示。

按照 $\varphi_{j,k}$ 的定义，将 $k=0$ 代入公式（13.22），式（13.35）可以改写如下



$$2^{-\frac{j}{2}}\varphi(2^{-j}t)=\sum_n p_n 2^{\frac{-j+1}{2}}\varphi(2^{-j+1}t-n) \tag{13.36}$$

把这个公式应用到哈尔函数的话，由图 13−10 可见，下面的关系式成立：

$$\varphi_{1,0}=2^{-\frac{1}{2}}(\varphi_{0,0}+\varphi_{0,1}) \tag{13.37}$$

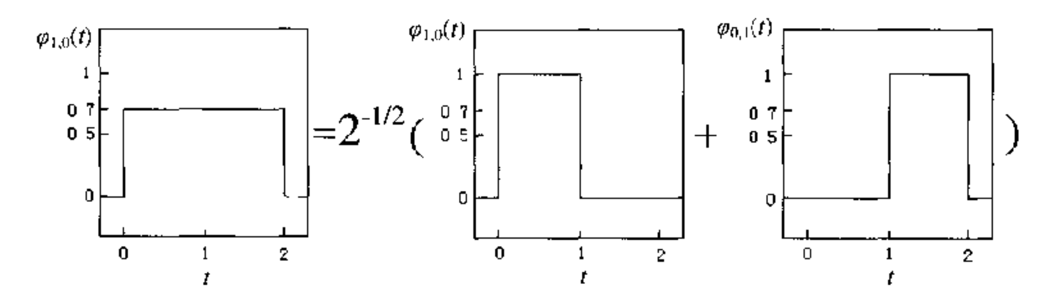


图 13-10

上述关系相当于在式（13.35）中，系数 $p_0=p_1=2^{-1/2}$ 的情况。这表明哈尔的第 1 级尺度函数能够用精度高一级的第 0 级尺度函数展开。

下面，为了考虑式（13.36）中 $k=0$ 以外的情况，把 t 用 $t-2^jk$ 置换，得：

$$\begin{aligned}2^{-\frac{j}{2}}\varphi(2^{-j}t-k)&=\sum_n p_n 2^{\frac{-j+1}{2}}\varphi(2^{-j+1}(t-2^jk)-n)\\&=2^{\frac{-j+1}{2}}\sum_n p_n\varphi(2^{-j+1}t-2k-n)\end{aligned} \tag{13.38}$$

此时，式（13.38）的左边就是定义式（13.22）的 $\varphi_{j,k}$，右边相当于式（13.22）中把 j 换成 $j-1$、k 换成 $2k+n$ 的情况，所以式（13.38）可以重写为：

$$\begin{aligned}\varphi_{j,k}(t)&=\sum_n p_n\varphi_{j-1,2k+n}(t)\\&=\sum_n p_{n-2k}\varphi_{j-1,n}(t)\end{aligned} \tag{13.39}$$

式（13.39）最后的式子，是把前面式子中的 n 置换成了 $n-2k$。

由式（13.39）可知，两边的 φ 是 j 的函数，但 p_n 不依赖于 j。也就是说，在展开中利用了与 j 的级数无关的相同序列 p_n。可以说，序列 p_n 是连接第 j 级尺度函数 $\varphi_{j,k}(t)$ 与精度高一级的第 j-1 级尺度函数 $\varphi_{j-1,k}(t)$ 的固有序列。

然而，根据多分辨率分析的定义，与尺度函数相同，第 j 级的小波 $\psi_{j,k}$ 也必须能够用第 $j-1$ 级尺度函数 $\varphi_{j-1,k}$ 展开。从而，与尺度函数的情况相同，下面的数学表达式成立：



$$\psi_{j,k}(t)=\sum_{n}q_{n-2k}\varphi_{j-1,n}(t) \tag{13.40}$$

其中，序列 q_n 是展开系数。这种情况也可以说序列 q_n 是连接第 j 级小波 $\psi_{j,k}(t)$ 与精度高一级的第 $j-1$ 级尺度函数 $\varphi_{j-1,n}(t)$ 的固有序列。由于式（13.39）和式（13.40）表示了 j 和 $j-1$ 两级尺度函数的关系以及尺度函数和小波的关系，所以被称为双尺度关系（two—scale relation）。

由图 13—11 可知，在哈尔小波中下面的关系式成立

$$\psi_{1,0}=2^{-\frac{1}{2}}(\varphi_{0,0}-\varphi_{0,1})$$

这种情况相当于式（13.40）中的 q_n 为 $q_0=2^{-1/2}$、$q_1=-2^{-1/2}$。

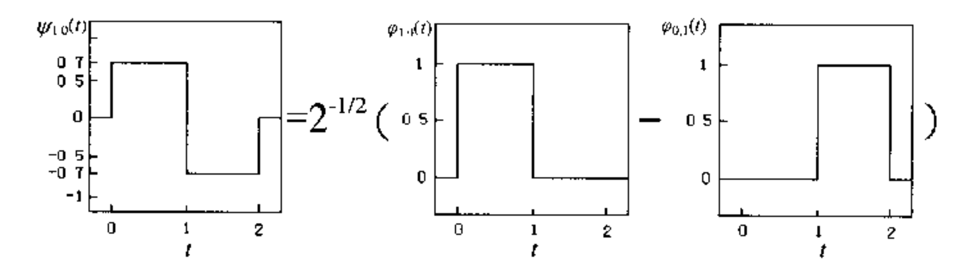
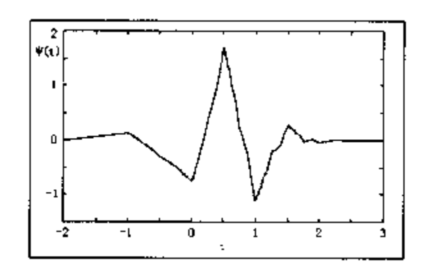


图 13-11

到目前为止，我们清楚了尺度函数是多分辨率分析所必需的。在满足了双尺度关系式（13.39）的条件以后，再根据另一个双尺度关系式（13.40），就可以求得对应于这个尺度函数的小波了。

除了哈尔小波以外，还有一些正交小波。例如，Daubechies 小波就是其中之一。Daubechies 小波及其尺度函数的形状复杂，用已知的函数难以表现。在 Daubechies 小波中采用自然数 N 来赋予小波特征。图 13—12 表示了 N=3 的 Daubechies 小波及其尺度函数，其中，(a)是小波，(b)是尺度函数。

(a) | (b)



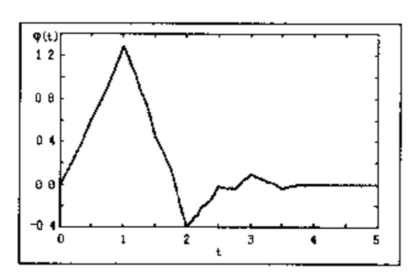


图 13-12



这里的 N 是 Daubechies 小波的特有表现，在序列 p_n 中表示非 0 要素的个数是 $2N$。换句话说，N 表示展开项数的 1/2。

Daubechies 小波也有如下特征

- 小波及其尺度函数在时间轴上的长度局限在 $2N-1$ 的范围内。即这个小波及其尺度函数具有紧支撑（compact support）。
- 在小波中，从 0 阶到 $N-1$ 阶的所有的矩皆为 0。

其中从 0 阶到 $N-1$ 阶的矩用公式表示的话，如下式所示：

$$\int_{-\infty}^{\infty} x^n \psi(x) \mathrm{d}x = 0 \qquad (n = 0, 1, 2, \cdots, N-1) \tag{13.41}$$

Daubechies 小波及其序列 p_k、q_k 如何被推导出来的？这是一个很有价值的课题。由于本书以应用为主，所以不予以论述，感兴趣的读者请参阅小波变换的专业书籍。

以后各节中，需要的序列 p_k、q_k 事先给出，如表示 Daubechies 尺度函数的序列 p_k 在表 13-1 中列出。

表 13-1 Daubechies 的序列 p_k

N=2	N=3	N=4
		0 2303778133088964
	0 3326705529500825	0.7148465705529154
0.4829629131445341	0 8068915093110924	0 6308807679298587
0.8365163037378077	0.4598775021184914	−0 0279837694168599
0.2241438680420134	−0.1350110200102546	−0 1870348117190931
0 1294095225512603	−0 0854412738820267	0.0308413818355607
	0 0352262918857095	0.0328830116668852
		−0 0105974017850690

表示小波序列的 q_k，是将 p_k 在时间轴方向上反转后，再将其系数符号反转得到的，即：

$$q_k = (-1)^k p_{-k} \tag{13.42}$$

然而，比较图 13-12 和表 13-1 中的 N=3 项会发现，表中的 p_k 仅定义了 6 个数值，而图却表示了一个相当复杂的函数形状。虽然本书并不讨论为什么仅 $2N$ 个数值却能够表现如此复杂的函数这个问题，但是通过重复迭代计算，从 $2N$ 个数值开始是可以顺次求出精度高的函数的。

在此简单介绍一下前述的式（13.10）所定义的 Daubechies 的离散小波变换表示法与 j 前面无负号的 Meyer 表示法的区别。研究者不同所使用的表示法也不一致，这两种表示法在使用上没有统一性，需要根据用途区别使用，这是小波变换最容易令初学者搞混的。也就是说，如果用于信号分析，考虑把被采样的信号作为最高分辨率的级，即 j=0，最低分辨率的级为 j=J，



这样，顺次从高分辨率到低分辨率，以级数 $j=0,1,2,\cdots,J$ 来表示更适当些。这种表示方法，根据采样级数，分辨率由高到低如 $j=-1,-2,\cdots$ 那样定义，这就是 Daubechies 表示法。另一方面，从表现小波的序列 p_k、q_k 开始，基于插值从低分辨率起顺次生成高分辨率的函数，随着分辨率的提高级数自然增加，这就是 Meyer 表示法。由于本书着眼于小波变换在图像处理中的应用，所以采用 Daubechies 表示法。

正如已经介绍的那样，Daubechies 小波与哈尔小波不同，不是直接给出 $\psi(t)$，而是仅给出 $2N$ 个离散序列。从而，由离散序列所表现的小波系数按式（13.18）求内积，首先从 $2N$ 个离散序列由重复迭代计算求出近似小波的高精度函数，接着，由数值计算求出信号与其函数的内积，这样的求法非常复杂。在下节中将说明从 $2N$ 个离散序列直接求出展开系数的方法。

13.5 离散小波变换

由上面的讨论可知，由 Daubechies 小波所代表的正交小波及其尺度函数可以用离散序列表示。在这一节中，利用这个离散序列来表示求小波展开系数的方法。这个方法是由 Mallat 在 1989 年发现的。

在基于计算机的小波变换中，信号作为连续波形是不能处理的，所以采用经一定间隔采样后的（有限）离散序列 $f(n)$。

如 13.3 节所述，连续信号 $f(t)$ 的第 0 级的近似函数 $f_0(t)$ 按照下式由第 0 级的尺度函数展开：

$$f(t) \approx f_0(t) = \sum_k s_k^{(0)} \varphi(t-k) \tag{13.43}$$

其中：

$$s_k^{(0)} = \int_{-\infty}^{\infty} f(t)\varphi_{0,k}^*(t)\mathrm{d}t \tag{13.44}$$

然而，在 Daubechies 小波中，虽然 $2N$ 个离散序列已经给出，但是由于尺度函数 $\varphi_{0,k}(t)$ 没有给出，所以存在用上式不能计算 $s_k^{(0)}$ 的问题。

为了克服这个问题，由 Mallat 提出的方法是：把对信号采样得到的序列 $f(n)$ 看作 $s_k^{(0)}$。Daubechies 小波如图 13－12 所示，哈尔小波如图 13－5 所示，Mallat 发现，由于 $\varphi_{0,k}$ 在矩形或三角形的窗口上改变 k、平移时间轴，所以对于某 k 值，$s_k^{(0)}$ 给出从窗口能够看到的范围的信号中间值。这个信号的中间值相当于 $f(k)$。这意味着 $\varphi_{0,k}$ 是像 $\delta_k(t)$ 那样的德尔塔函数（δ function）。Mallat 认为 $\varphi_{0,k}(t)$ 具有基于德尔塔函数 $\delta_k(t)$ 重构相同的作用。下面来看一下哈尔小波的例子。图 13－13 的最上图为信号 $f(t)$，采样间隔为 1 秒，把 1 秒间隔的平均值作为采样值时的 $f(n)$ 如中间图所示。另外，$s_k^{(0)}$ 按式（13.21）给出，如最下图所示。

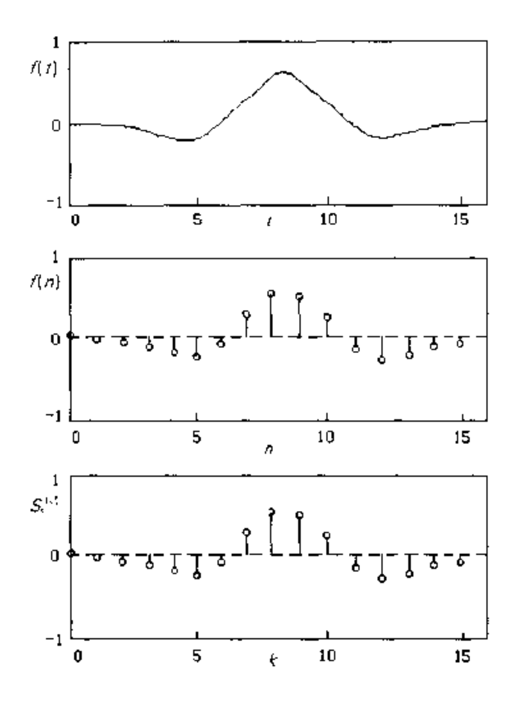
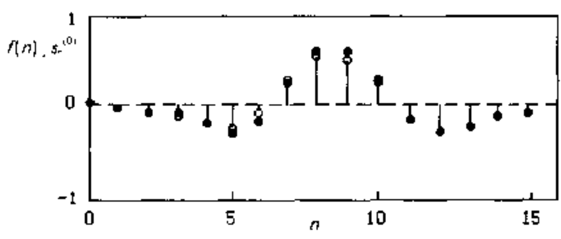


图 13-13

$f(n)$和 $s_k^{(0)}$按照哈尔尺度函数的定义式（13.20）是相同的，即 $f(n)=s_k^{(0)}$。对于哈尔以外的小波，一般说 $f(n)\neq s_k^{(0)}$，但是在图像处理的应用方面 $f(n)$看作 $s_k^{(0)}$被证明实用上是没有问题的。

作为一个例子，由数值计算所求得的 Daubechies 尺度系数 $s_n^{(0)}$（N=2）与 $f(n)$的比较结果如图 13—14 所示，从图中可以看到 $f(n)$和 $s_k^{(0)}$几乎没有什么区别。



其中，白点表示信号采样值 $f(n)$，黑点表示基于数值计算的 Daubechies 的第 0 级的尺度系数（n=2）

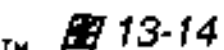


图 13-14



既然已经得到了 $s_k^{(0)}$，那么，就可以基于 $s_k^{(0)}$ 求第 0 级以外的尺度系数 $s_k^{(j)}$ 及小波系数 $w_k^{(j)}$ 了。

如式（13.25）所定义的那样，尺度系数由下式给出：

$$s_k^{(j)}=\int_{-\infty}^{\infty}f(t)\varphi_{j,k}^*(t)\mathrm{d}t \tag{13.45}$$

利用双尺度关系式（13.39），把上式的 $\varphi_{j,k}(t)$ 用 $\varphi_{j-1,n}(t)$ 表示的话，得到下面的关系式

$$\begin{aligned}s_k^{(j)}&=\int_{-\infty}^{\infty}f(t)\sum_n p_{n-2k}^*\varphi_{j-1,n}^*(t)\mathrm{d}t\\&=\sum_n p_{n-2k}^*\int_{-\infty}^{\infty}f(t)\varphi_{j-1,n}^*(t)\mathrm{d}t\end{aligned} \tag{13.46}$$

上式的积分部分一定是 $s_n^{(j-1)}$，所以上式重写后得：

$$s_k^{(j)}=\sum_n p_{n-2k}^*s_n^{(j-1)} \tag{13.47}$$

通过使用这个公式，能够从第 0 级的尺度系数 $s_k^{(0)}$，依次求出高级数（低分辨率）的尺度系数。

那么，下面从 $s_k^{(j-1)}$ 求级数高一级（分辨率低一级）的小波系数 $w_k^{(j)}$。由式（13.18）所定义的那样，小波系数由下式给出

$$w_k^{(j)}=\int_{-\infty}^{\infty}f(t)\psi_{j,k}^*(t)\mathrm{d}t \tag{13.48}$$

利用双尺度关系式（13.39），把上式的 $\psi_{j,k}(t)$ 用 $\psi_{j-1,n}(t)$ 表示的话，得到下面的关系式：

$$\begin{aligned}w_k^{(j)}&=\int_{-\infty}^{\infty}f(t)\sum_n q_{n-2k}^*\varphi_{j-1,n}^*(t)\mathrm{d}t\\&=\sum_n q_{n-2k}^*\int_{-\infty}^{\infty}f(t)\varphi_{j-1,n}^*(t)\mathrm{d}t\end{aligned} \tag{13.49}$$

上式的积分部分就是 $s_n^{(j-1)}$，所以上式重写后得：

$$w_k^{(j)}=\sum_n q_{n-2k}^*s_n^{(j-1)} \tag{13.50}$$

通过使用这个公式，能够从第 0 级的尺度系数 $s_k^{(0)}$ 依次求出高级数（低分辨率）的小波系数。

由式（13.47）可求得第 j−1 级的尺度系数 $s_k^{(j-1)}$ 与第 j 级的尺度系数 $s_k^{(j)}$ 的关系，而由式（13.50）可求得第 j−1 级的尺度系数 $s_k^{(j-1)}$ 与第 j 级的小波系数 $w_k^{(j)}$ 的关系。如果考虑用式（13.31）把第 j−1 级的近似函数 $f_{j-1}(t)$ 分解成第 j 级的近似函数 $f_j(t)$ 和小波成分 $g_j(t)$ 的话，那么尺度函数将一个接一个地分解成低精度（低分辨率）的尺度系数和小波系数，其分解状态如图 13−15 所示。



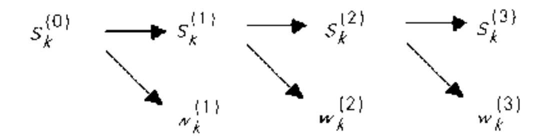


图 13-15

按照图 13-7 的例子，算出 Daubechies 小波（N=2），如图 13-16 所示，其中上图表示信号的采样值 $f(n)$，中间图表示第 1 级的尺度系数 $s_k^{(1)}$，最下图表示第 1 级的小波系数 $w_k^{(1)}$。在图 13-7 中表示了近似函数 $f_1(t)$，与之相对，这里表示了尺度系数。可见两者具有类似的形状。

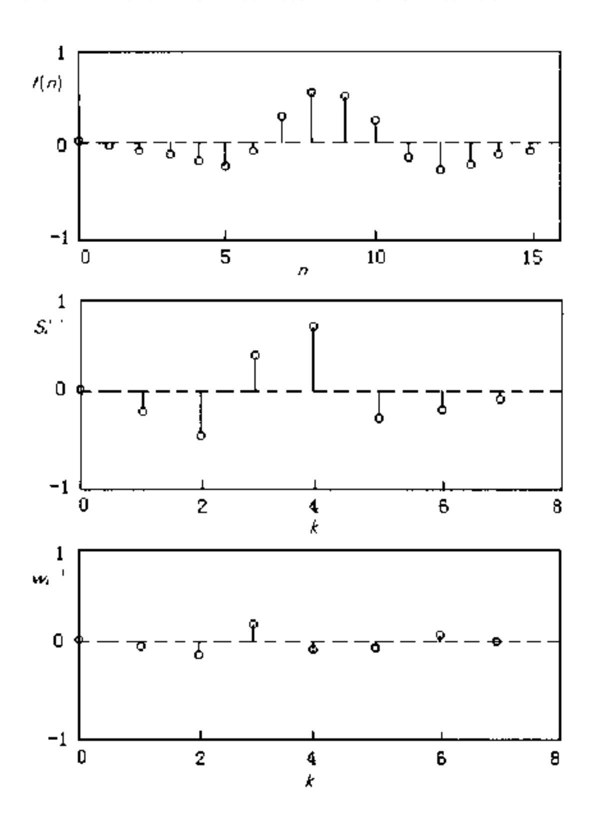


图 13-16

下面，对从低精度（低分辨率）的小波系数及尺度系数，求出高精度（高分辨率）的尺度系数的方法进行说明。由式（13.17）可知，第 j-1 级的近似函数 f_{j-1} 可用下式表示：



$$f_{j-1}(t) = f_j(t) + g_j(t) = \sum_k s_k^{(j)} \varphi_{j,k}(t) + \sum_k w_k^{(j)} \psi_{j,k}(t) \tag{13.51}$$

另一方面，由式（13.26）可知，$s_k^{(j-1)}$可由第 j−1 级的近似函数和尺度函数的内积来求出，即

$$s_k^{(j-1)} = \int_{-\infty}^{\infty} f_{j-1}(t)\varphi_{j-1,k}^*(t)\mathrm{d}t \tag{13.52}$$

把式(13.51)代入上式，得，

$$\begin{aligned} s_n^{(j-1)} &= \int_{-\infty}^{\infty} (\varphi_{j-1,n}^*(t)\sum_k s_k^{(j)}\varphi_{j,k}(t) + \varphi_{j-1,n}^*(t)\sum_k w_k^{(j)}\psi_{j,k}(t))\mathrm{d}t \\ &= \sum_k [s_k^{(j)}\int_{-\infty}^{\infty}\varphi_{j-1,n}^*(t)\varphi_{j,k}(t)\mathrm{d}t + w_k^{(j)}\int_{-\infty}^{\infty}\varphi_{j-1,n}^*(t)\psi_{j,k}(t)\mathrm{d}t] \end{aligned} \tag{13.53}$$

其中，依据式(13.39)和尺度函数的平移规范正交条件（式(13.23)），可得：

$$\begin{aligned} \int_{-\infty}^{\infty} \varphi_{j-1,n}^*(t)\varphi_{j,k}(t)\mathrm{d}t &= \int_{-\infty}^{\infty}\varphi_{j-1,n}^*(t)\sum_l p_{l-k}\varphi_{j-1,l}(t)\mathrm{d}t \\ &= \sum_l p_{l-2k}\int_{-\infty}^{\infty}\varphi_{j-1,n}^*(t)\varphi_{j-1,l}(t)\mathrm{d}t \\ &= p_{n-2k} \end{aligned} \tag{13.54}$$

在此，上式右边的积分利用了 $n=l$ 以外的情况都为 0 的正交条件。

同样，依据式(13.40)和尺度函数的平移规范正交条件，可得到下式：

$$\int_{-\infty}^{\infty} \varphi_{j-1,n}^*(t)\psi_{j,k}(t)\mathrm{d}t = q_{n-2k} \tag{13.55}$$

综上所述，可得到下面的关系式(13.56)。按照这个关系式从第 j 级的小波系数及尺度系数能够求得精度高一级的第 j−1 级的尺度系数。

$$s_n^{(j-1)} = \sum_k [p_{n-2k}s_k^{(j)} + q_{n-2k}w_k^{(j)}] \tag{13.56}$$

根据这个重构公式，对于 j 依次重复迭代这一过程，从而求得高精度级数的尺度系数。这个重构过程如图 13−17 所示。

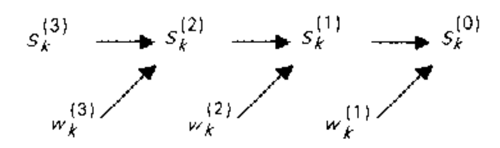


图 13-17



与基于数值计算用内积求小波系数及尺度系数的方法（式(13.25)及式(13.18)）相比，由于式（13.47）及式（13.50）的计算量少，计算速度快，因此被称为快速小波变换（fast wavelet transform）或者 Mallat 人字形算法（Mallat's herringbone algorithm）。

13.6 二维离散小波变换

这一节对使用离散小波的二维图像数据的小波变换进行说明。图像数据作为二维的离散数据给出，用 $f(m, n)$表示。与二维离散傅立叶变换的情况相同，首先进行水平方向上的离散小波变换，对其系数再进行垂直方向上的小波变换。与上节相同，把图像数据 $f(m,n)$看作第 0 级的尺度系数 $s_{m,n}^{(0)}$。

首先，进行水平方向上的离散小波变换。

$$\begin{aligned} s_{m,n}^{(j+1,x)} &= \sum_k p_{k-2m}^* s_{k,n}^{(j)} \\ w_{m,n}^{(j+1,x)} &= \sum_k q_{k-2m}^* s_{k,n}^{(j)} \end{aligned} \tag{13.57}$$

其中，$s_{m,n}^{(j+1,x)}$及 $w_{m,n}^{(j+1,x)}$分别表示水平方向的尺度系数及小波系数。j=0 时如图 13—18 所示。

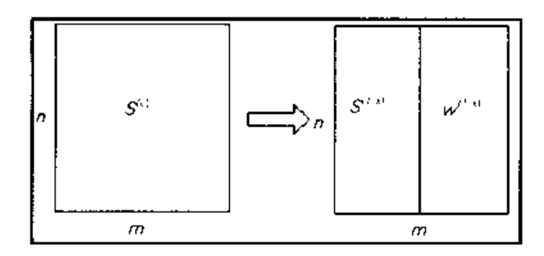


图 13-18

接着，分别对系数进行垂直方向的离散小波变换。

$$\begin{aligned} s_{m,n}^{(j+1)} &= \sum_l p_{l-2n}^* s_{m,l}^{(j+1,x)} \\ w_{m,n}^{(j+1,h)} &= \sum_l q_{l-2n}^* s_{m,l}^{(j+1,x)} \\ w_{m,n}^{(j+1,v)} &= \sum_l p_{l-2n}^* w_{m,l}^{(j+1,x)} \\ w_{m,n}^{(j+1,d)} &= \sum_l q_{l-2n}^* w_{m,l}^{(j+1,x)} \end{aligned} \tag{13.58}$$



其中，$w_{m,n}^{(j+1,h)}$表示在水平方向上使尺度函数起作用、垂直方向上使小波起作用的系数，$w_{m,n}^{(j+1,v)}$表示在水平方向上使小波起作用、垂直方向上使尺度函数起作用的系数，另外，$w_{m,n}^{(j+1,d)}$表示在水平和垂直方向上全部使小波起作用的系数。j=0 时如图 13—19 所示。

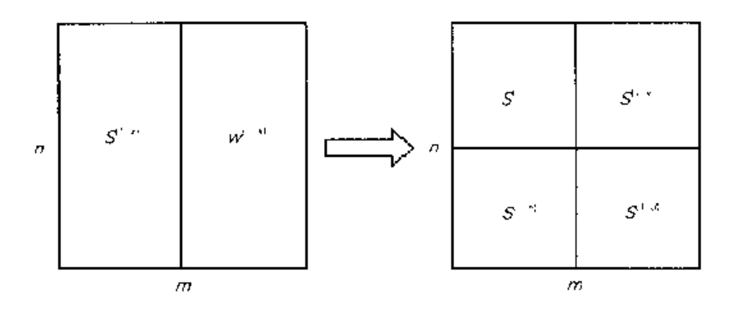


图 13-19

综合式（13.57）和式（13.58）得：

$$
\begin{aligned}
s_{m,n}^{(j+1)} &= \sum_{l}\sum_{k} p_{k-2m}^{*} p_{l-2n}^{*} s_{k,l}^{(j)} \\
w_{m,n}^{(j+1,h)} &= \sum_{l}\sum_{k} p_{k-2m}^{*} q_{l-2n}^{*} s_{k,l}^{(j)} \\
w_{m,n}^{(j+1,v)} &= \sum_{l}\sum_{k} q_{k-2m}^{*} p_{l-2n}^{*} s_{k,l}^{(j)} \\
w_{m,n}^{(j+1,d)} &= \sum_{l}\sum_{k} q_{k-2m}^{*} q_{l-2n}^{*} s_{k,l}^{(j)}
\end{aligned}
\tag{13.59}
$$

上式中仅对$s_{m,n}^{(j+1)}$再进一步分解成 4 个成分，通过不断重复迭代这一过程，可以进行多分辨率分解。

这个重构与一维的情况相同，按下式进行：

$$
\begin{aligned}
s_{m,n}^{(j)} = \sum_{k}\sum_{l} [& p_{m-2k} p_{n-2l} s_{k,l}^{(j+1)} + p_{m-2k} q_{n-2l} w_{k,l}^{(j+1,h)} \\
& + q_{m-2k} p_{n-2l} w_{k,l}^{(j+1,v)} + q_{m-2k} q_{n-2l} w_{k,l}^{(j+1,d)}]
\end{aligned}
\tag{13.60}
$$

13.7　图像的小波变换

List 13.1 和 List 13.2 表示了一维离散小波变换和其逆变换的程序。List 13.3 和 List 13.4 表示了二维离散小波变换和其逆变换的程序。这些程序都是按照第 j 级的尺度系数 $s^{(j)}$来求第 j+1 级的小波系数 $w^{(j+1)}$及尺度系数 $s^{(j+1)}$的，接着，由 $w^{(j+1)}$及 $s^{(j+1)}$重构 $s^{(j)}$。小波采用了 Daubechies（N=2）方式。表示这个小波的尺度函数的序列 p_k在表 13—2 中给出。

另外，表示小波的序列 q_k，是将 p_k 在时间轴方向上反转后，再将其系数符号反转得到的，如式（13.42）所示。

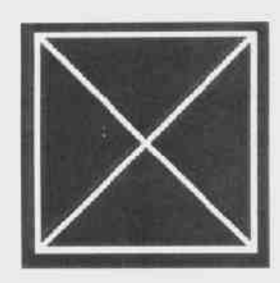
表 13-2　Daubechies 序列 p_k

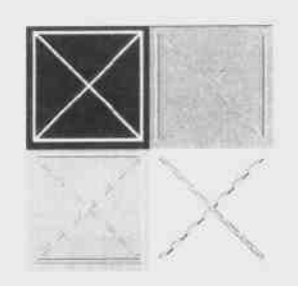
N=2	N=3	N=4
		0.23037781330889
	0.33267055295008	0.71484657055291
0.48296291314453	0.80689150931109	0.63088076792986
0.83651630373780	0.45987750211849	−0.02798376941686
0.22414386804201	−0.13501102001025	−0.18703481171909
−0.12940952255126	−0.08544127388203	0.03084138183556
	0.03522629188571	0.03288301166689
		−0.01059740178507
		0.02667005790055
		0.18817680007763
	0.05441584224311	0.52720118893158
	0.31287159091432	0.68845903945344
0.11154074335011	0.67563073629732	0.28117234366057
0.49462389039845	0.58535468365422	−0.24984642432716
0.75113390802110	−0.01582910525638	−0.19594627437729
0.31525035170920	−0.28401554296158	0.12736934033575
−0.22626469396544	0.00047248457391	0.09305736460355
−0.12976686756727	0.12874742662049	−0.07139414716635
0.09750160558732	−0.01736930100181	−0.02945753682184
0.02752286553031	−0.04408825393080	0.03321267405936
−0.03158203931749	0.01398102791740	0.00360655356699
0.00055384220116	0.00874609404741	−0.01073317548330
0.00477725751095	−0.00487035299345	0.00139535174707
−0.00107730108531	−0.00039174037338	0.00199240529519
	0.00067544940645	−0.00068585669496
	−0.00011747678412	−0.00011646685513
		0.00009358867032
		−0.00001326420289

我们可以使用 List 13.5 的程序对图 13−20（a）所示的图像进行二维小波变换。List 13.5 使用 List 13.3 的 wavelet2d 函数对图像信号 $f(m,n)$ 进行分解，使用 List 13.4 的 iwavelet2d 函数进行重构。13.20（b）表示对原始图像信号分解（即第 1 级小波分解）后的 4 个成分（或称为 4 个子图像），由图可见，$w^{(1,v)}$ 表现垂直方向上的高频成分，$w^{(1,h)}$ 表现水平方向上的高频成分，$w^{(1,d)}$ 表现对角线方向上的高频成分。另外，$s^{(1)}$ 表现对 $s^{(0)}$ 平均化的低频成分。对 $s^{(1)}$ 再进一步分解成 4 个成分（即第 2 级小波分解）的结果如图 13−20（c）所示。



(a) | (b)
(c)





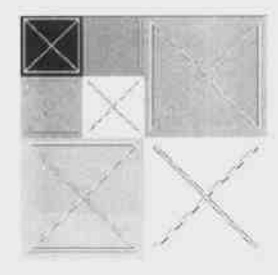


图 13-20

可见，在图像分解过程中，总的数据量既没有增加也没有减少。但是，一个图像经过小波变换后，得到一系列不同分辨率的子图像，即表示低频成分的子图像及表现不同方向上高频成分的子图像。高频成分的子图像上大部分数值都接近于 0，越是高频这种现象越明显。所以，对于一幅图像来说，包含图像主要信息的是低频成分，而高频成分仅包含细节信息。因此，一个最简单的图像压缩方法是保存低频成分而丢掉高频成分。

图 13–21 表示了小波分解和压缩的例子。图 13–21（b）是对图 13–21（a）所示原始图像进行第 1 级小波分解后的结果，图 13–21（c）表示只利用 1 级分解后的低频成分（左上角的子图像）进行图像恢复的结果，图 13–21（d）表示对图 13–21（b）中的低频成分的子图像再进行第 2 级小波分解后的结果，最后对第 2 级分解后的低频成分（左上角的子图像）进行恢复后的图像如图 13–21（e）所示。

可见，保留低频成分的压缩方法虽然简单，但是在图像压缩后没有细节信息，影响图像效果。在互联网上传输图像可以使用这种方法，首先传送低分辨率（高级数）的图像，然后再传送分辨率高一级的图像，直到最高分辨率的图像。这样能够产生渐进的效果，首先呈现图像的大体轮廓，然后再逐渐更细致地呈现图像，如同图像越来越近，越来越清晰。

小波图像压缩的另一种方法是利用小波树，在此不作介绍。

小波变换不仅在图像压缩方面，而且在图像处理的其他方面也有广泛的应用。小波变换在图像处理领域的应用可归纳为以下两个方面：



- 展开系数的操作和重构——基于展开系数的操作的图像数据压缩、消除噪声、特定图像模式的增强、退化图像的复原、不可视信息的嵌入（图像水印）等。
- 相似相关的检测——基于小波与相似图像模式的检测，如癌细胞等的特定模式的检测、缺陷检测、图像检索、纹理（表面模样）的分类、纹理的区域分割等。

限于篇幅，在此将结束本书的介绍。

(a)(b)(c)
(d)(e)



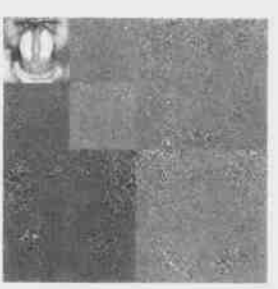




图 13-21

List 13.1　一维小波变换

```
#include "StdAfx.h"
#include "BaseList.h"

#include <stdio.h>
#include <math.h>
//#include "fwt.h"

/*--- Wavelet1d ----一维小波变换-----------------------------------------
     s0       输入信号
     s_len    输入信号长
     p        比例配列 pk
```



```
    q         小波配列 qk
    sup       配列长
    s1        分解信号 s1
    w1        分解信号 w1
----------------------------------------------------------------*/
void Wavelet1d (double *s0, int s_len,  double *p, double *q, int sup,
       double *s1, double *w1)
{
    int n, k;
    int index;

    for (k = 0; k < s_len/2; k++) {
    s1[k] = 0.0;
    w1[k] = 0.0;
    for (n = 0; n < sup; n++) {
    index = (n+2*k)%s_len ;
       s1[k] += p[n] * s0[index];
       w1[k] += q[n] * s0[index];
       }
    }
}
```

List 13.2 一维逆小波变换

```
#include "StdAfx.h"
#include "BaseList.h"

#include <stdio.h>
#include <math.h>
//#include 'fwt.h"

/*---Iwavelete1d----维逆小波变换-------------------------------------------
   s1        分解信号 s1
   w1        分解信号 w1
   s_len     信号长
   p         小波数列 pk
   q         比例数列 qk
```



```
    sup      数列长
    s0       恢复信号
------          -----------------------      -----------------------*/
void Iwavelet1d (double *s1, double *w1, int s_len, double *p,
        double *q, int sup, double *s0)
{
    int n, k;
    int index,ofs;

    ofs=max(1024,s_len);  //为了不使 index 为负数的补正值
    for (n = 0; n < s_len; n++) {
        s0[2*n+1] = 0.0;
        s0[2*n] = 0.0;
        for (k = 0; k < sup/2; k++) {
            index = (n-k+ofs)%s_len;
            s0 [2*n+1] += p[2*k+1] * s1[index] + q[2*k+1] * w1[index];
            s0 [2*n] += p[2*k] * s1[index] + q[2*k] * w1[index];
        }
    }

}
```

List 13.3　二维小波变换

```
#include "StdAfx.h"
#include "BaseList.h"

#include <stdio.h>
#include <math.h>

/*--- Wavelete2d --- 二维小波变换-----------------------------------------------
     image_in:   输入图像数据指针
     xsize:      图像宽度
     ysize:      图像高度
     s1          j+1 级的二维比例系数
     w1v         j+1 级的二维小波系数（垂直方向成分）
     w1h         j+1 级的二维小波系数（水平方向成分）
```



```
  wld        j+1 级的二维小波系数（对角方向成分）
---------------------------------------------------------------*/
void Wavelet2d (BYTE *image_in, int xsize, int ysize,
       double *s1, double *w1v, double *w1h, double *w1d)
{

    if(xsize != ysize) return;

    int i,j;
    double *s1x, *w1x;
    double *s1xt, *w1xt;
    double *s1t, *w1ht, *w1vt, *w1dt;
    double *s0;    // j 级的二维比例系数
    int s_len; //输入信号长
    s_len = xsize;

    s0 = new double[xsize*ysize];

    s1x = new double[s_len*(s_len/2)];
    w1x = new double[s_len*(s_len/2)];

    s1xt = new double[(s_len/2)*s_len];
    w1xt = new double[(s_len/2)*s_len];

    s1t = new double[(s_len/2)*(s_len/2)];
    w1ht = new double[(s_len/2)*(s_len/2)];
    w1vt = new double[(s_len/2)*(s_len/2)];
    w1dt = new double[(s_len/2)*(s_len/2)];

    int sup = 4; //小波数列 p_k 的长度
    double p[4]  = {0.482962913145, 0.836516303738,
            0.224143868042, -0.129409522551};
             //分割数列 p_k (N=2)
    double q[4];  //分割数列 q_k (N=2)

    for(i=0;i<sup;i++)  //由 p_k 生成 q_k
       q[i]=pow(-1,i)*p[sup-i-1];
```



```
for(j = 0; j < ysize; j++)
for(i=0; i<xsize; i++)
*(s0 + j*xsize + i) = (double)(*(image_in + j*xsize +i));

for (j = 0; j < s_len; j++) {
    Wavelet1d(&(*(s0 +j*s_len)),s_len,p,q,sup, &(*(s1x + j*(s_len/2))),
  &(*(w1x + j*(s_len/2))));
} //将 s0 分解为 s1x 和 w1x

for(j=0;j<s_len;j++){  //行列倒置
  for(i=0;i<s_len/2;i++){
      *(s1xt + i*s_len + j )=*(s1x + j*(s_len/2) + i);
      *(w1xt + i*s_len + j )=*(w1x + j*(s_len/2) + i);
  }
}

for (j = 0; j < s_len/2; j++) {
Wavelet1d( &(*(s1xt + j*s_len)),s_len,p,q,sup, &(*(s1t + j*(s_len/2))),
  &(*(w1ht + j*(s_len/2))));
     //将 s1x 分解为 s1 和 w1h
Wavelet1d(&(*(w1xt+j*s_len)),s_len,p,q,sup,&(*(w1vt + j*(s_len/2))),
  &(*(w1dt + j*(s_len/2))));
     //将 w1x 分解为 w1v 和 w1d
}

for(j=0;j<s_len/2;j++){  //行列倒置
    for(i=0;i<s_len/2;i++){
      *(s1 + i*(s_len/2) + j)=*(s1t + j*(s_len/2) + i);
      *(w1h + i*(s_len/2) + j)=*(w1ht + j*(s_len/2) + i);
      *(w1v + i*(s_len/2) + j)=*(w1vt + j*(s_len/2) + i);
      *(w1d + i*(s_len/2) + j)=*(w1dt + j*(s_len/2) + i);
    }
}

delete [] s1x;
delete [] w1x;
```



```
    delete [] s1xt;
    delete [] w1xt;

    delete [] s1t;
    delete [] w1ht;
    delete [] w1vt;
    delete [] w1dt;
    delete [] s0;
  }
```

List 13.4 二维逆小波变换

```
#include "StdAfx.h"
#include "BaseList.h"

#include <stdio.h>
#include <math.h>

/*--- Iwavelete2d --- 二维逆小波变换-------------------------------------
    s1           j+1 级的二维比例系数
    w1v          j+1 级的二维小波系数（垂直方向成分）
    w1h          j+1 级的二维小波系数（水平方向成分）
    w1d          j+1 级的二维小波系数（对角方向成分）
    image_out:   输出图像数据指针
    xsize:       图像宽度
    ysize:       图像高度


------------------------------------------------------------------*/

void Iwavelet2d (double *s1, double *w1v, double *w1h, double *w1d,
       BYTE *image_out, int xsize, int ysize)
{

    if(xsize != ysize) return;
    int i,j;
    double *s1x, *w1x;
```



```
double *s1xt, *w1xt;
double *s1t, *w1ht, *w1vt, *w1dt;
double *s0;      // j 级的二维比例系数
int s_len;       //输入信号长
s_len = xsize/2;

s0 = new double[xsize*ysize];

s1x = new double[(2*s_len)*s_len];
w1x = new double[(2*s_len)*s_len];

s1xt = new double[s_len*(2*s_len)];
w1xt = new double[s_len*(2*s_len)];

s1t = new double[s_len*s_len];
w1ht = new double[s_len*s_len];
w1vt = new double[s_len*s_len];
w1dt = new double[s_len*s_len];

int sup = 4; //小波数列 p_k 的长度
double p[4] = {0.482962913145, 0.836516303738,
           0.224143868042, -0.129409522551};
             //分割数列 p_k (N=2)
double q[4];     //分割数列 q_k (N=2)

for(i=0;i<sup;i++)  //由 p_k 生成 q_k
    q[i]=pow(-1,i)*p[sup-i-1];

for(j=0;j<s_len;j++){  //行列倒置
  for(i=0;i<s_len;i++){
      *(s1t + j*s_len + i) = *(s1 + i*s_len + j);
      *(w1ht + j*s_len + i) = *(w1h + i*s_len + j);
      *(w1vt + j*s_len + i) = *(w1v + i*s_len + j);
      *(w1dt + j*s_len + i) = *(w1d + i*s_len + j);
  }
}
```



```
for (j = 0; j < s_len; j++) {
Iwavelet1d(&(*(s1t + j*s_len)),&(*(w1ht + j*s_len)),s_len,p,q,sup,
  &(*(s1xt + j*2*s_len)));
      //由 s1 和 w1h 恢复 s1x
    Iwavelet1d(&(*(w1vt + j*s_len)),&(*(w1dt + j*s_len)),s_len,p,q,sup,
  &(*(w1xt + j*2*s_len)));
      //由 w1v 和 w1d 恢复 w1x
}

for(j=0;j<s_len;j++){ //行列倒置
  for(i=0;i<s_len*2;i++){
*(s1x + i*s_len + j) = *(s1xt + j*2*s_len + i );
    *(w1x + i*s_len + j) = *(w1xt + j*2*s_len + i );
  }
}

//由 s1x 和 w1x 恢复 s0
for (j = 0; j < s_len*2; j++) {
Iwavelet1d(&(*(s1x+j*s_len)),&(*(w1x+j*s_len)),s_len,p,q,sup,&(*(s0 +j*2*s_len)));
}

double value;
for(j = 0; j < ysize; j++)
{
for(i=0; i < xsize; i++)
{
  value = *(s0 + j*xsize + i);
  *(image_out + j*xsize +i) = (BYTE)value;
  *(image_out + j*xsize +i) = (BYTE)(*(s0 + j*xsize + i));
}
}
delete [] s1x;
delete [] w1x;

delete [] s1xt;
delete [] w1xt;
```



```
        delete [] s1t;
        delete [] w1ht;
        delete [] w1vt;
        delete [] w1dt;
        delete [] s0;

    }
```

List 13.5　二维小波信号图像化

```
#include "StdAfx.h"
#include "BaseList.h"

#include <stdio.h>
#include <math.h>

/*---Wavelete2d_image ---二维小波信号图像化------------------------------
    s1          j+1 级的二维比例系数
    w1v         j+1 级的二维小波系数（垂直方向成分）
    w1h         j+1 级的二维小波系数（水平方向成分）
    w1d         j+1 级的二维小波系数（对角方向成分）
    image_out:  输出图像数据指针
    xsize:      图像宽度
    ysize:      图像高度

-----------------------------------------------------------------------*/
//二维小波信号图像化
void Wavelet2d_image (double *s1, double *w1v, double *w1h, double *w1d,
            BYTE *image_out, int xsize, int ysize)
{
    int i,j;
    double value;

    double maxs = (double)0;
    double mins = (double)500;

    double maxv = (double)0;
    double minv = (double)500;
```



```
double maxh = (double)0;
double minh = (dcuble)500;

double maxd = (double)0;
double mind = (double)500;

double sizes, sizev, sizeh, sized;

for(j = 0; j < ysize/2; j++ )
{
  for(i = 0; i < xsize/2; i++ )
  {
        value = *(s1 + j*xsize/2 + i);
    if(value > maxs ) maxs = value;
    if(value < mins ) mins = value;

    value = *(w1v + j*xsize/2 + i);
    if(value > maxv ) maxv = value;
    if(value < minv ) minv = value;

    value = *(w1h + j*xsize/2 + i);
    if(value > maxh ) maxh = value;
    if(value < minh ) minh = value;

    value = *(w1d + j*xsize/2 + i);
    if(value > maxd ) maxd = value;
    if(value < mind ) mind = value;

  }
}

sizes = maxs - mins;
sizev = maxv - minv;
sizeh = maxh - minh;
sized = maxd - mind;
for(j = 0; j < ysize/2; j++ )
```



```
{
  for(i = 0; i < xsize/2; i++ )
  {
        value = *(s1 + j*xsize/2 + i);
     *(image_out+j*xsize+i)=(BYTE)((value-mins)*(double)255/ sizes);

     value = *(w1v + j*xsize/2 + i);
     *(image_out + j*xsize + i+xsize/2)=(BYTE)( (value-minv)*(double)255 / sizev);

     value = *(w1h + j*xsize/2 + i);
     *(image_out+(j+ysize/2)*xsize + i) =(BYTE)( (value-minv)*(double)255 / sizeh);

     value = *(w1d + j*xsize/2 + i);
     *(image_out+(j+ysize/2)*xsize + i+xsize/2) = (BYTE)((value-minv)*
     (double)255/ sized);

   }
 }
}
```



展开表节点，分别将 Orders 和 Order Details 拖到窗体 Form1 中，自动生成 SqlConnection1 和 SqlDataAdapter1 与 SqlDataAdapter1 对象。

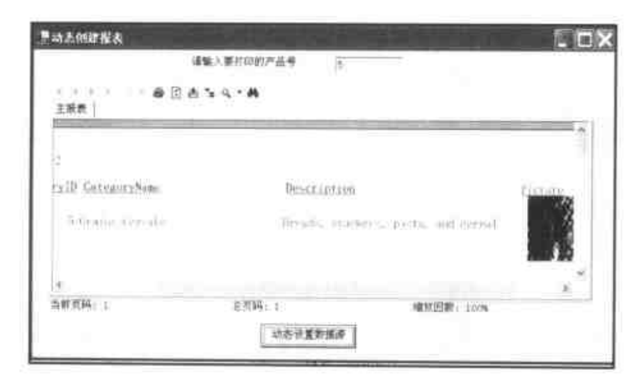


图 11-33

4 添加一个数据集 Dataset1，将 Orders 和 Order Details 加入到数据集中。

5 按照前面介绍的方法，添加一个水晶报表 CrystalReport1，使用报表专家，插入表 Orders 和 Order Details，在“链接”选取项卡中使用 OrderID 字段链接，在“字段”选项卡中选择要显示的主表和明细表的字段，在“组”选项卡中分组依据选择为 Orders 表的 OrdersID 字段，在“总计”选项卡的汇总字段中选择 Order Details.Quantity，汇总类型选择“求和”，设置完后，单击“完成”按钮。

6 在报表设计器中调整需要显示的字段的位置、宽度等。

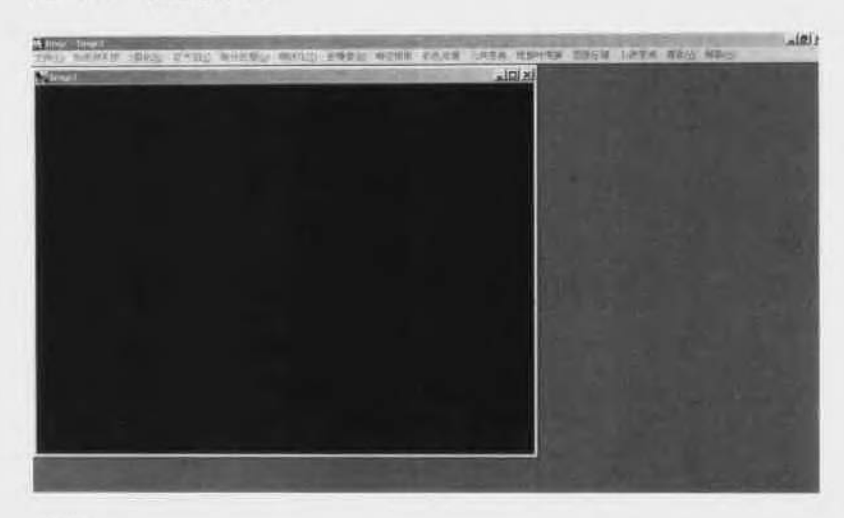
7 在窗体 Form1 的 Load 事件过程中写入代码：

```
Dim con As New SqlConnection ("server=localhost;database=Northwind;_
integrated security=true")
Dim adapter1 As New SqlDataAdapter ("select * from orders where orderid=_
10248", con)
Dim adapter2 As New SqlDataAdapter ("select * from [order details]", con)
Dim ds As New DataSet ()
adapter1.Fill (ds, "order")
adapter1.Fill (ds, "orderdetails")
Dim oprt As New CrystalReport1 ()
oprt.SetDataSource (ds.Tables (0) )
CrystalReportViewer1.ReportSource = oprt
```

5 / 用程序改变报表中 text 的文本

如果想在程序中通过代码改变Text中的值，可使用下列代码：

附 1.1　系统界面



附图 1

附 1.1.1　系统表示源程序

```
// ImgcView.cpp : implementation of the CImgcView class
//

#include "stdafx.h"
#include "Imgc.h"

#include "ImgcDoc.h"
#include "ImgcView.h"
#include "Cdib.h"
#include "Global.h"
#include "BaseList.h"

#include "HistDlg.h"
#include "BinaryDlg.h"
#include "CAbstractDlg.h"
#include "ColorDlg.h"
#include "ColAbstractDlg.h"
```



```
#include "SynthDlg.h"
#include "DistortionDlg.h"
#include "FftDlg.h"
#include "DpcmDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////
// CImgcView

IMPLEMENT_DYNCREATE(CImgcView, CScrollView)

BEGIN_MESSAGE_MAP(CImgcView, CScrollView)
 //{{AFX_MSG_MAP(CImgcView)
   ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
   ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
   ON_COMMAND(ID_FILE_SAVE, OnFileSave)
   ON_COMMAND(ID_HIST, OnHist)
   ON_COMMAND(ID_THESHOLD, OnTheshold)
   ON_COMMAND(ID_DIFFERENTIAL, OnDifferential)
   ON_COMMAND(ID_THINNING, OnThinning)
   ON_COMMAND(ID_NOISE_SMOOTH, OnNoiseSmooth)
   ON_COMMAND(ID_NOISE_MEDIAN, OnNoiseMedian)
   ON_COMMAND(ID_NOISE_BINARY_ERO, OnNoiseBinaryEro)
   ON_COMMAND(ID_NOISE_BINARY_DIL, OnNoiseBinaryDil)
   ON_COMMAND(ID_CHARACTER_ABSTRACT, OnCharacterAbstract)
   ON_COMMAND(ID_COLOR_BAR, OnColorBar)
   ON_COMMAND(ID_COLOR_TRANSFORM, OnColorTransform)
   ON_COMMAND(ID_COLOR_ABSTRACT, OnColorAbstract)
   ON_COMMAND(ID_COLOR_SYNTH, OnColorSynth)
   ON_COMMAND(ID_DISTORTION, OnDistortion)
   ON_COMMAND(ID_FFT, OnFft)
   ON_COMMAND(ID_COLOR_TO_MONO, OnColorToMono)
```



```
    ON_COMMAND(ID_IMAGE_DPCM, OnImageDpcm)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CImgcView construction/destruction

CImgcView::CImgcView()
{
    // TODO: add construction code here

//初始化图像窗口的宽度
      m_nXSize = 640;

      //初始化图像窗口的高度
      m_nYSize = 480;

      // 初始化图像数据指针
      m_pImage = NULL;
      m_pImageR = NULL;
      m_pImageG = NULL;
      m_pImageB = NULL;

      m_bFileOpen = FALSE;

}

CImgcView::~CImgcView()
{
      //解散图像数据内存
      if(m_pImage != NULL)
 {
   delete m_pImage;
```



```
    m_pImage = NULL;
  }
  if(m_pImageR != NULL)
  {
    delete m_pImageR;
    m_pImageR = NULL;
  }
  if(m_pImageG != NULL)
  {
    delete m_pImageG;
    m_pImageG = NULL;
  }
  if(m_pImageB != NULL)
  {
    delete m_pImageB;
    m_pImageB = NULL;
  }

  ::DeleteDispDib();

  delete m_pDifferenDlg;
}

BOOL CImgcView::PreCreateWindow(CREATESTRUCT& cs)
{
  // TODO: Modify the Window class or styles here by modifying
  //  the CREATESTRUCT cs

  return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CImgcView drawing

void CImgcView::OnDraw(CDC* pDC)
{
   CImgcDoc* pDoc = GetDocument();
```



```
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CDib *pDib ;
    HGDIOBJ hOrg ;
    CPalette *pOrgPal ;

    ////////////////////////////////////////////
    // Get current image
    pDib = ::GetDib() ;

    /////
    hOrg = ::SelectObject(m_cMemoryDC.m_hDC, m_hBitmap) ;

    // Draw image
    pDib->UsePalette(&m_cMemoryDC) ;
    pDib->Draw(&m_cMemoryDC, CPoint(0, 0), CSize(GetXSize(), GetYSize())) ;

    // set paletter
    pOrgPal=(CPalette*)pDC->SelectPalette(pDC->GetCurrentPalette(),FALSE) ;
    // view
    pDC->BitBlt(0, 0, GetXSize(), GetYSize(), &m_cMemoryDC, 0, 0, SRCCOPY);
    // recover paletter
    pDC->SelectPalette(pOrgPal, FALSE) ;

    ////////////////////////////////////////////
    // release DIB
    ::SelectObject(m_cMemoryDC.m_hDC, hOrg) ;
}

void CImgcView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    //设定滚动轴的大小
    SetWindowSize(m_nXSize, m_nYSize);

      /////// initialize
```



```
    CClientDC dc(this);

    ::CreateDispDib(&dc, m_nXSize, m_nYSize);

    CDib *pDib ;
    pDib = ::GetDib() ;

    //表示当前图像
    // made DIB bitmap
    if ((m_hBitmap = ::CreateDIBitmap(dc.m_hDC, pDib->m_lpBMIH, 0L, NULL, NULL,
    0)) != 0) {
      // initialize structure data by GetDIBits()
      ::GetDIBits(dc.m_hDC, m_hBitmap, 0, m_nYSize, NULL, (LPBITMAPINFO)(pDib->
      m_lpBMIH), DIB_RGB_COLORS);
      // made memory DC
      m_cMemoryDC.CreateCompatibleDC(&dc) ;
    }
    ////////////////////////////////////////////

    m_pDifferenDlg = new CDifferenDlg();

    m_bFileOpen = FALSE;

    Invalidate();
}

/////////////////////////////////////////////////////////////////////////////
// CImgcView printing

BOOL CImgcView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CImgcView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
```

```
    // TODO: add extra initialization before printing
}
void CImgcView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CImgcView diagnostics

#ifdef _DEBUG
void CImgcView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CImgcView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

CImgcDoc* CImgcView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CImgcDoc)));
    return (CImgcDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CImgcView message handlers

void CImgcView::OnFileOpen()
{
 //读入位图文件
 Load_imagefile_bmp();

    //获得图像大小
```



```
    m_nXSize = ::GetXSize();
    m_nYSize = ::GetYSize();
    //设定图像窗口大小
    SetWindowSize(m_nXSize, m_nYSize);

//更新画面
Invalidate();

//判断图像格式
if( ::GetImageType() == 24)
{
    //消除老图像数据
    if(m_pImageR != NULL)
    {
      delete[] m_pImageR;
      m_pImageR = NULL;
    }
    if(m_pImageG != NULL)
    {
      delete[] m_pImageG;
      m_pImageG = NULL;
    }
    if(m_pImageB != NULL)
    {
      delete[] m_pImageB;
      m_pImageB = NULL;
    }

    //为新图像分配内存
    m_pImageR = new BYTE[m_nXSize*m_nYSize];
    m_pImageG = new BYTE[m_nXSize*m_nYSize];
    m_pImageB = new BYTE[m_nXSize*m_nYSize];

    //读入新图像数据
    ::ReadImageDataRGB(m_pImageR, m_pImageG, m_pImageB);
}
    else if(::GetImageType() == 8)
```



```
    //消除原始图像数据
    if(m_pImage != NULL)
    {
      delete[] m_pImage;
      m_pImage = NULL;
    }

    //为新图像分配内存
    m_pImage = new BYTE[m_nXSize*m_nYSize];

    //读入新图像数据
    ::ReadImageData(m_pImage);
  }

 m_bFileOpen = TRUE;

    //更新画面
    Invalidate();

}

void CImgcView::SetWindowSize(int xsize, int ysize)
{
    //设定滚动轴的大小
    SetScrollSizes(MM_TEXT, CSize(xsize-1, ysize-1));

    /////////初始化表示帧的大小
    CRect cRect ;
    CMDIChildWnd* pChildFrm = (CMDIChildWnd*)GetParentFrame();

// 改变表示的大小
    cRect.SetRect(0, 0, xsize, ysize) ;
    CalcWindowRect((LPRECT)cRect, CWnd::adjustOutside) ;
    SetWindowPos(&wndTopMost,cRect.left,cRect.top,cRect.Width(), cRect.
    Height(), SWP_NOZORDER|SWP_NOMOVE) ;
```



```
    // 改变帧的大小
    pChildFrm->CalcWindowRect((LPRECT)cRect) ;
    pChildFrm->SetWindowPos(NULL, cRect.left, cRect.top, cRect.Width(),cRect.
    Height(), SWP_NOZORDER|SWP_NOMOVE) ;
    ////////////////////////////////////////////////
}

void CImgcView::OnFileSave()
{
    if(!m_bFileOpen)//没有读入图像
      return;
    ::Save_imagefile_bmp();

}

void CImgcView::OnFileSaveAs()
{
    if(!m_bFileOpen)//没有读入图像
      return;

    ::SaveAs_imagefile_bmp();
}

void CImgcView::OnHist()
{
    if(!m_bFileOpen)//没有读入图像
      return;

    int type = ::GetImageType();

    if(type != 8)//非灰度图像
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    CHistDlg dlg(this);
```



```
    dlg.DoModal();

}

void CImgcView::OnTheshold()
{
    //没有读入图像退出
if(!m_bFileOpen)
    return;

    int type = ::GetImageType();

    if(type != 8)//非灰度图像
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    CBinaryDlg dlg(this);
    dlg.DoModal();

}

void CImgcView::OnDifferential()
{
    //没有读入图像退出
if(!m_bFileOpen)
    return;

    int type = ::GetImageType();

    if(type != 8)//非灰度图像
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }
```



```
    m_pDifferenDlg->Create(this);

}

void CImgcView::OnThinning()
{
    //没有读入图像退出
    if(!m_bFileOpen)
    return;

    int type = ::GetImageType();

    //非灰度图像退出
    if(type != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

 //定义结果图像指针
    BYTE *image_thin;

 //分配内存
    image_thin = new BYTE[m_nXSize*m_nYSize];

    //读入表示的图像数据
    ::ReadImageData(m_pImage);

    //细线化处理
    ::Thinning(m_pImage, image_thin, m_nXSize, m_nYSize);

    //表示处理结果
    ::Disp_image(image_thin);

    //解放内存
    delete image_thin;
```



```
    //更新画面
    Invalidate();
}

void CImgcView::OnNoiseSmooth()
{
    //非灰度图像退出
    if(!m_bFileOpen)//没有读入图像
      return;

    //获得图像类型（彩色、灰度）
    int type = ::GetImageType();

    //非灰度图像退出
    if(type != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    //定义结果图像指针
    BYTE *image_smooth;

    //分配内存
    image_smooth = new BYTE[m_nXSize*m_nYSize];

    //读入表示的图像数据
    ::ReadImageData(m_pImage);

    //移动平均法去噪音
    ::Image_smooth(m_pImage, image_smooth, m_nXSize, m_nYSize);

    //表示处理结果
    ::Disp_image(image_smooth);

    //解放内存
    delete image_smooth;
```



```
    //更新画面
    Invalidate();

}

void CImgcView::OnNoiseMedian()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
      return;

    //获得图像种类（彩色、灰度）
    int type - ::GetImageType();

    //非灰度图像时退出
    if(type != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    //定义结果图像指针
    BYTE *image_median;

    //分配内存
    image_median = new BYTE[m_nXSize*m_nYSize];

    //读入表示的图像数据
    ::ReadImageData(m_pImage);

    //中值法去噪音
    ::Median(m_pImage, image_median, m_nXSize, m_nYSize);

    //表示处理结果
    ::Disp_image(image_median);

    //解放内存
```



```
    delete image_median;
    //更新画面
    Invalidate();
}

void CImgcView::OnNoiseBinaryEro()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
      return;

    //获得图像种类（彩色、灰度）
    int type = ::GetImageType();

    //非灰度图像时退出
    if(type != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    //定义结果图像指针
    BYTE *image_ero;

    //分配内存
    image_ero = new BYTE[m_nXSize*m_nYSize];

    //读入表示的图像数据
    ::ReadImageData(m_pImage);

    //二值图像腐蚀
    ::Erodible(m_pImage, image_ero, m_nXSize, m_nYSize);

    //表示处理结果
    ::Disp_image(image_ero);

    //解放内存
```



```
    delete image_ero;
    //更新画面
    Invalidate();
}

void CImgcView::OnNoiseBinaryDil()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
      return;

    //非灰度图像时退出
    if(::GetImageType() != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    //定义结果图像指针
    BYTE *image;

    //分配内存
    image = new BYTE[m_nXSize*m_nYSize];

    //读入表示的图像数据
    ::ReadImageData(m_pImage);

    //二值图像膨胀
    ::Dilation(m_pImage, image, m_nXSize, m_nYSize);

    //表示处理结果
    ::Disp_image(image);

    //解放内存
    delete image;

    //更新画面
```



```
    Invalidate();
}

void CImgcView::OnCharacterAbstract()
{
    //没有读入图像时退出
if(!m_bFileOpen)
        return;

    //非灰度图像时退出
    if(::GetImageType() != 8)
    {
      AfxMessageBox("该命令只能处理灰度图像。");
      return;
    }

    //定义对象
    CCAbstractDlg dlg(this);

    //打开模式窗口
dlg.DoModal();

}

void CImgcView::OnColorBar()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
    AfxMessageBox("请预先读入彩色图像。");
    return;
    }

    //非彩色图像时退出
    if(::GetImageType() != 24)
    {
      AfxMessageBox("请读入一幅彩色图像。");
```



```
        return;
    }

    //定义结果图像指针
    BYTE *image_r;
    BYTE *image_g;
    BYTE *image_b;

    //获得图像大小
    int xsize = ::GetXSize();
    int ysize = ::GetYSize();

    //分配内存
    image_r = new BYTE[xsize*ysize];
    image_g = new BYTE[xsize*ysize];
    image_b = new BYTE[xsize*ysize];

    //做彩条
    ::Colorbar(image_r, image_g, image_b, xsize, ysize, 255);

    //图像表示
    ::Disp_imageRGB(image_r, image_g, image_b);

    //解放内存
    delete image_r;
    delete image_g;
    delete image_b;

    //更新画面
    Invalidate();
}

void CImgcView::OnColorTransform()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
```



```
        AfxMessageBox("请预先读入彩色图像。");
        return;
    }

    //非彩色图像时退出
    if(::GetImageType() != 24)
    {
        AfxMessageBox("请读入一幅彩色图像。");
        return;
    }

    //定义对象
    CColorDlg dlg(this);

    //打开模式窗口
dlg.DoModal();
}

void CImgcView::OnColorAbstract()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
        AfxMessageBox("请预先读入彩色图像。");
        return;
    }

    //非彩色图像时退出
    if(::GetImageType() != 24)
    {
        AfxMessageBox("请读入一幅彩色图像。");
        return;
    }

    //定义对象
    CColAbstractDlg dlg(this);
```



```
    //打开模式窗口
dlg.DoModal();

}

void CImgcView::OnColorSynth()
{
    //定义对象
    CSynthDlg dlg(this);

    //打开模式窗口
dlg.DoModal();

}

void CImgcView::OnDistortion()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
      AfxMessageBox("请预先读入灰度图像。");
      return;
    }

    //非灰度图像时退出
    if(::GetImageType() != 8)
    {
      AfxMessageBox("请读入一幅灰度图像。");
      return;
    }

    //定义对象
    CDistortionDlg dlg(this);

    //打开模式窗口
dlg.DoModal();
}
```



```
void CImgcView::OnFft()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
      AfxMessageBox("请预先读入灰度图像, ");
      return;
    }

    //非灰度图像时退出
    if(::GetImageType() != 8)
    {
      AfxMessageBox("请读入一幅灰度图像。");
      return;
    }

     AfxMessageBox("注意  图像的长和宽必须是2的次方。");

    //定义对象
    CFftDlg dlg(this);

    //打开模式窗口
    dlg.DoModal();
}

void CImgcView::OnColorToMono()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
      AfxMessageBox("请预先读入彩色图像, ");
      return;
    }

    //非彩色图像时退出
    if(::GetImageType() != 24)
```



```
        return;
    //彩色图像转灰度图像
    ::Color_to_mono();

    //获得图像大小
    m_nXSize = ::GetXSize();
    m_nYSize = ::GetYSize();

    //解放旧内存
    if(m_pImage != NULL)
    {
      delete [] m_pImage;
      m_pImage = NULL;
    }

    //分配新内存
    m_pImage = new BYTE[m_nXSize*m_nYSize];

    //读入灰度图像数据
    ::ReadImageData(m_pImage);

    //更新画面
    Invalidate();
}


void CImgcView::OnImageDpcm()
{
    //没有读入图像时退出
    if(!m_bFileOpen)
    {
      AfxMessageBox("请预先读入灰度图像。");
      return;
    }

    //非灰度图像时退出
    if(::GetImageType() != 8)
```



```
    {
      AfxMessageBox("请读入一幅灰度图像。");
      return;
    }

    //定义对象
    CDpcmDlg dlg(this);

    //打开模式窗口
dlg.DoModal();
}
```

附 1.1.2 系统表示头文件

```
// ImgcView.h : interface of the CImgcView class
//
//////////////////////////////////////////////////////////////////////

#if!defined(AFX_IMGCVIEW_H__FAE6E9A5_B318_41A3_BFA3_FFA787B13626__INCLUDED_)
#define AFX_IMGCVIEW_H__FAE6E9A5_B318_41A3_BFA3_FFA787B13626__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "DifferenDlg.h"
class CImgcView : public CScrollView
{
protected: // create from serialization only
   CImgcView();
   DECLARE_DYNCREATE(CImgcView)

// Attributes
public:
   CImgcDoc* GetDocument();

// Operations
public:
```



```
    //图像窗口的宽度
    int m_nXSize;

    //图像窗口的高度
int m_nYSize;
    //图像数据(灰度)
BYTE *m_pImage;

    //图像数据(R)
    BYTE *m_pImageR;

    //图像数据(G)
    BYTE *m_pImageG;

    //图像数据(B)
    BYTE *m_pImageB;

    //判断图像是否读入参数
    BOOL m_bFileOpen;

    //图像表示用
    HBITMAP m_hBitmap;
    CDC m_cMemoryDC ;

    //改变图像窗口函数
    void SetWindowSize(int xsize, int ysize);

    //微分处理窗口对象指针
    CDifferenDlg *m_pDifferenDlg;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CImgcView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
```



```
    virtual void OnInitialUpdate(); // called first time after construct
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CImgcView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CImgcView)
    afx_msg void OnFileOpen();
    afx_msg void OnFileSaveAs();
    afx_msg void OnFileSave();
    afx_msg void OnHist();
    afx_msg void OnTheshold();
    afx_msg void OnDifferential();
    afx_msg void OnThinning();
    afx_msg void OnNoiseSmooth();
    afx_msg void OnNoiseMedian();
    afx_msg void OnNoiseBinaryEro();
    afx_msg void OnNoiseBinaryDil();
    afx_msg void OnCharacterAbstract();
    afx_msg void OnColorBar();
    afx_msg void OnColorTransform();
    afx_msg void OnColorAbstract();
    afx_msg void OnColorSynth();
    afx_msg void OnDistortion();
    afx_msg void OnFft();
```



```
    afx_msg void OnColorToMono();
    afx_msg void OnImageDpcm();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in ImgcView.cpp
inline CImgcDoc* CImgcView::GetDocument()
   { return (CImgcDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
// !defined(AFX_IMGCVIEW_H__FAE6E9A5_B318_41A3_BFA3_FFA787B13626__INCLUDED_)
```

附 1.2　阈值处理



附图 2

附 1.2.1　阈值处理源程序

```
// BinaryDlg.cpp : implementation file
//

#include "stdafx.h"
```



```
#include "Imgc.h"
#include "BinaryDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBinaryDlg dialog

CBinaryDlg::CBinaryDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CBinaryDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CBinaryDlg)
   m_nThresh = 100;
   m_nDown = 0;
   m_nThreshH = 120;
   //}}AFX_DATA_INIT

   //指针初始化
   m_pImageO = NULL;
   m_pImageB = NULL;
   m_pImageR = NULL;

     //保存表示画面句柄
   m_pParent = pParent;
}
void CBinaryDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CBinaryDlg)
   DDX_Text(pDX, IDC_THRESHOLD, m_nThresh);
   DDV_MinMaxInt(pDX, m_nThresh, 0, 255);
```



```
    DDX_Radio(pDX, IDC_THRESHOLD_RADIO1, m_nDown);
    DDX_Text(pDX, IDC_THRESHOLD_HIGH, m_nThreshH);
    DDV_MinMaxInt(pDX, m_nThreshH, 0, 255);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CBinaryDlg, CDialog)
     //{{AFX_MSG_MAP(CBinaryDlg)
     ON_BN_CLICKED(ID_BINARY, OnBinary)
     ON_BN_CLICKED(IDC_ORIGINAL_IMAGE, OnOriginalImage)
     ON_BN_CLICKED(IDC_THRESHOLD_RADIO3, OnThresholdRadio3)
     ON_BN_CLICKED(IDC_THRESHOLD_RADIO1, OnThresholdRadio1)
     ON BN_CLICKED(IDC_THRESHOLD_RADIO2, OnThresholdRadio2)
     ON_BN_CLICKED(IDC_BINARY_REVERSE, OnBinaryReverse)
     //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBinaryDlg message handlers
/////////////////

void CBinaryDlg::OnCancel()
{
     // TODO: Add extra cleanup here

     //解放内存
     if(m_pImageO)
     {
       delete[] m_pImageO;
       m_pImageO = NULL;
     }
     if(m_pImageB)
     {
       delete[] m_pImageB;
       m_pImageB = NULL;
     }
     if(m_pImageR)
```



```
    {
      delete[] m_pImageR;
      m_pImageR = NULL;
    }

    CDialog::OnCancel();
}

BOOL CBinaryDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //获得图像大小
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
    m_pImageO = new BYTE[m_xsize*m_ysize];
    m_pImageB = new BYTE[m_xsize*m_ysize];
    m_pImageR = new BYTE[m_xsize*m_ysize];

    //读入图像数据
    ::ReadImageData(m_pImageO);

    m_bOri = TRUE;

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CBinaryDlg::OnBinary()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    int model = 1;
```



```
    if(m_nDown == 0) model = 2;

    //双阈值二值化处理
    if(m_nDown == 2)
    {
      if(m_nThreshH < m_nThresh) m_nThreshH = m_nThresh;
      UpdateData(FALSE);
      ::Threshold_mid(m_pImageO, m_pImageB, m_xsize, m_ysize, m_nThresh,
      m_nThreshH);
    }

    //一般二值化处理
    else
        ::Threshold(m_pImageO, m_pImageB, m_xsize, m_ysize, m_nThresh, model);

    //表示二值图像
    ::Disp_image(m_pImageB);

    //更新画面
    m_pParent->Invalidate();

    m_bOri = FALSE;
}

void CBinaryDlg::OnBinaryReverse()
{
    // TODO: Add your control notification handler code here

    //反转数据
    if(m_bOri)
    ::Reverse_image(m_pImageO, m_pImageR, m_xsize, m_ysize);
    else ::Reverse_image(m_pImageB, m_pImageR, m_xsize, m_ysize);

    //表示反转后的图像
    ::Disp_image(m_pImageR);

    //更新画面
```



```
    m_pParent->Invalidate();
}
void CBinaryDlg::OnOriginalImage()
{
    // TODO: Add your control notification handler code here

    //表示原图像
    ::Disp_image(m_pImage0);

    //更新画面
    m_pParent->Invalidate();

    m_bOri = TRUE;
}

void CBinaryDlg::OnThresholdRadio3()
{
    // TODO: Add your control notification handler code here

    //高阈值编辑窗口有效
    GetDlgItem(IDC_THRESHOLD_HIGH)->EnableWindow(TRUE);
}

void CBinaryDlg::OnThresholdRadio1()
{
    // TODO: Add your control notification handler code here

    //高阈值编辑窗口无效
    GetDlgItem(IDC_THRESHOLD_HIGH)->EnableWindow(FALSE);
}

void CBinaryDlg::OnThresholdRadio2()
{
    // TODO: Add your control notification handler code here

    //高阈值编辑窗口无效
    GetDlgItem(IDC_THRESHOLD_HIGH)->EnableWindow(FALSE);
```



```
}
```

附 1.2.2　阈值处理头文件

```
#if!defined(AFX_BINARYDLG_H__16975377_9AD8_4B04_86A9_96885F307EE1__INCLUDED_)
#define AFX_BINARYDLG_H__16975377_9AD8_4B04_86A9_96885F307EE1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// BinaryDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBinaryDlg dialog

class CBinaryDlg : public CDialog
{
// Construction
public:
   CBinaryDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
   //{{AFX_DATA(CBinaryDlg)
   enum { IDD = IDD_BINARY_DLG };
   int    m_nThresh;
   int    m_nDown;
   int    m_nThreshH;
   //}}AFX_DATA

   //父亲窗口句柄
   CWnd* m_pParent;

   //图像尺寸大小
   int m_xsize;
   int m_ysize;

   //原图像数据指针
```



```
    BYTE *m_pImageO;
    //二值图像数据指针
    BYTE *m_pImageB;

    //反转图像数据指针
    BYTE *m_pImageR;

    BOOL m_bOri;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBinaryDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

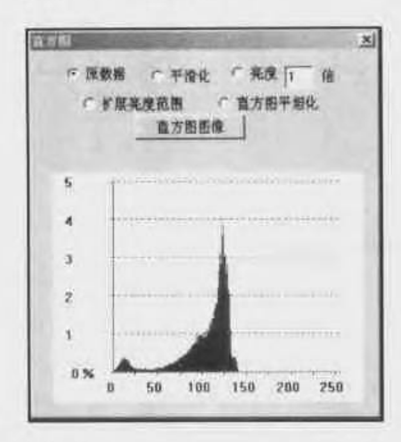
    // Generated message map functions
    //{{AFX_MSG(CBinaryDlg)
    virtual void OnCancel();
    virtual BOOL OnInitDialog();
    afx_msg void OnBinary();
    afx_msg void OnOriginalImage();
    afx_msg void OnThresholdRadio3();
    afx_msg void OnThresholdRadio1();
    afx_msg void OnThresholdRadio2();
    afx_msg void OnBinaryReverse();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
//Microsoft Visual C++ will insert additional declarations immediately before the
  previous line.
```



```
#endif
// !defined(AFX_BINARYDLG_H__16975377_9AD8_4B04_86A9_96885F307EE1__
INCLUDED_)
```

附 1.3 直方图



附图 3

附 1.3.1 直方图处理源程序

```
// HistDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "HistDlg.h"
#include "Global.h"
#include "BaseList.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////
// CHistDlg dialog
```



```
CHistDlg::CHistDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CHistDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CHistDlg)
   m_nHistType = 0;
   m_fn = 1.0f;
   //}}AFX_DATA_INIT

   //保存表示画面句柄
   m_pParent = pParent;
}

void CHistDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CHistDlg)
   DDX_Radio(pDX, IDC_HIST_ORIGINAL, m_nHistType);
   DDX_Text(pDX, IDC_HIST_N_EDIT, m_fn);
   DDV_MinMaxFloat(pDX, m_fn, 1.f, 100.f);
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CHistDlg, CDialog)
   //{{AFX_MSG_MAP(CHistDlg)
   ON_WM_PAINT()
   ON_BN_CLICKED(IDC_HIST_ORIGINAL, OnHistOriginal)
   ON_BN_CLICKED(IDC_HIST_SMOOTH, OnHistSmooth)
   ON_BN_CLICKED(IDC_HIST_AMPLIFY, OnHistAmplify)
   ON_BN_CLICKED(IDC_HIST_EXPAND, OnHistExpand)
   ON_BN_CLICKED(IDC_HIST_PLAN, OnHistPlan)
   ON_BN_CLICKED(IDC_HIST_IMAGE, OnHistImage)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHistDlg message handlers
```



```
BOOL CHistDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //描画移动量
m_shift_val = 30;

    m_left = 0;    //左
    m_top = 0;     //上
    m_right = 200; //右
    m_bottom = 100;//下

    //轴的数据表示
    m_bAxisText = TRUE;

    //获得图像尺寸
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
    m_pImage_in = new BYTE[m_xsize*m_ysize];
    m_pImage_out = new BYTE[m_xsize*m_ysize];
    m_pImage_hist = new BYTE[m_xsize*m_ysize];

    //读入图像数据
    ::ReadImageData(m_pImage_in);

    //计算直方图数列
    ::Histgram(m_pImage_in, m_xsize, m_ysize, m_hist);

    m_create = 1;

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```



```
void CHistDlg::PostNcDestroy()
{
    //解放内存
    delete[] m_pImage_in;
    delete[] m_pImage_out;
    delete[] m_pImage_hist;

    CDialog::PostNcDestroy();
}

void CHistDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // 获取绘制坐标的文本框
    CWnd* pWnd = GetDlgItem(IDC_HIST_STATIC);

    // 画面更新
    pWnd->Invalidate();
    pWnd->UpdateWindow();

    //获得窗口数据
    UpdateData(TRUE);

    //直方图的全体描画
    DispHistogram();

    // Do not call CDialog::OnPaint() for painting messages
}

void CHistDlg::DispHistogram()
{
    CDC* pDC;
    CRect rect;
    CWnd* pWnd = GetDlgItem(IDC_HIST_STATIC);
    pWnd->GetClientRect( &rect );
```



```
    //左
m_left = rect.left + 2;

//上
 m_top = rect.top + 8;

    //右
    m_right = rect.right - 2;

    //下
    m_bottom = rect.bottom - 2;

    m_shift_val = 30;

    //获得资源的关系
    pDC = pWnd->GetDC();

    //确定窗口参数
    UpdateData(TRUE);

    //用平滑后数据计算百分比
    if(m_nHistType == 1)
      ::CalHistPercent(m_histSmooth, m_disp_array, m_max_percent);

    //用原数据计算百分比
else
      ::CalHistPercent(m_hist, m_disp_array, m_max_percent);

    //设定 y 轴最大值
    m_max_value_y = (int)(m_max_percent+1);

    //范围定为 0~100
    if( m_max_value_y >= 100 ) m_max_value_y = 100;
    if( m_max_value_y <= 0 )   m_max_value_y =   0;

    //参数、描画区域设定
    shortret = DrawHistArea(pDC);
```



```
    if( ret == OK ){

    //x、y 轴的描画
    DrawAxisLine(pDC);

    //x、y 轴文字的表示
    DrawAxisText(pDC);

    //亮度值的表示
    DrawGrayValueMono(pDC);
    }

    //资源关系的解放
    pWnd->ReleaseDC(pDC);
}

short CHistDlg::DrawHistArea(CDC* pDC)
{
    int i, max_value, min_value;

    //x 轴
    //内部放大率
    //n 倍表示基准幅
m_inzoom_x  =  (float)1;

//x 轴的小刻度
    m_small_x  =  (float)5;

    //x 轴的大刻度
    m_large_x  =  (float)25;

    //x 轴文字表示刻度的变化量
    m_text_delta_x  =  25;
    //x 轴的最大值
    m_max_value_x  =  255;

    //y 轴
```



```
    // 5, 10, 25, 50, 100
max_value = 100;
min_value = 100/2;

    //求最大比例
    for( i=0 ; i<5 ; i++ ){
      if( (min_value < m_max_value_y)&&( m_max_value_y <= max_value ) ){
        break;
      }
      else{
        max_value = min_value;
        if( min_value < 5 ){ min_value = 2; break; }
        if( min_value -- 25 ) min_value = 10;
        else         min_value = min_value/2;
      }
    }
    m_max_value_y = max_value;

    m_text_delta_y = m_max_value_y / 5;//1/5
    if(m_text_delta_y == 0) m_text_delta_y = 1;
    m_large_y = (float)m_text_delta_y; //==
    m_small_y = m_large_y / (float)10;//1/10

    //设定内部扩大率
    if( m_max_value_y == 0 ) m_inzoom_y = (float)1;

    //内部扩大率
    else m_inzoom_y =  100 / (float)m_max_value_y ;

    //
    //右上领域的宽度、高度  m_resize_x, m_resize_y
    m_resize_x = (float)(m_right - m_left);
    m_resize_y = (float)(m_bottom - m_top);
    //x 界限
    if( m_resize_x < 200 ) return(NG);
    if( m_shift_val < 30 ) return(NG);
```



```
    //y 界限
    if( m_resize_y < 100 ) return(NG);

    //基准长 x
    m_zoom_x = ( m_resize_x - (float)60 ) / (float)280;

    //基准长 y
    m_zoom_y =  m_resize_y  / (float) 120;

    //矩形区域描画
    CPen  Pen(PS_SOLID, 1, RGB(255,255,255) );
    CPen* pOldPen = pDC->SelectObject(&Pen);
    pDC->Rectangle( m_left, m_bottom+1,
                    m_left+(int)m_resize_x+1,
                    m_bottom-(int)m_resize_y );
    pDC->SelectObject(pOldPen);

    return(OK);
}

//x、y 的刻度文字表示
void CHistDlg::DrawAxisText( CDC* pDC )
{
 int i;
 float old_posi_y, new_posi_y;
 float old_posi_x, new_posi_x;

 CString str_x, str_y;

 pDC->SetTextAlign( TA_LEFT | TA_BOTTOM );
 pDC->SetTextColor( RGB(0,0,0) );

 //文字拥挤时拉开距离表示
    if( m_text_delta_y != 0 )
    {
        old_posi_y = (float)(m_bottom-20);
```



```
      //y 的刻度文字表示
      str_y.Format("%3s", "0 %" );

      // "0%"  文字的微调整
      pDC->TextOut( m_left+17, (int)old_posi_y, str_y, 3 );

      for( i=1; i<(int)(m_max_value_y/m_text_delta_y+1) ; i++ ){
      new_posi_y= (float)m_bottom-(float)20-(m_text_delta_y*(float)i*
      m_zoom_y*m_inzoom_y);

      //y 轴数字间的宽度
      //m_text_interval_y=16
      if( (old_posi_y - (float)16) > new_posi_y ){

        //y 轴刻度数字
        str_y.Format("%3d", m_text_delta_y*i );
        pDC->TextOut( m_left+3, (int)new_posi_y, str_y, 3 );
        old_posi_y = new_posi_y;
      }
    }
  }

 if( m_text_delta_x != 0 ){
   old_posi_x = (float)(m_left+m_shift_val*2 - 15);

   //x 轴刻度数字
   str_x.Format("%3d", m_text_delta_x*0 );

   pDC->TextOut( (int)old_posi_x, m_bottom-5, str_x, 3 );

   for( i=1; i<(int)(m_max_value_x/m_text_delta_x+1) ; i++ ){
     new_posi_x = (float)m_left+(float)m_shift_val*2
       +(m_text_delta_x*(float)i*m_zoom_x*m_inzoom_x)-(float)15;
     //x 轴文字间隔
     if(  (new_posi_x - old_posi_x) > (float)28){

       //x 轴刻度数值
```



```
            str_x.Format("%3d", m_text_delta_x*i );
            pDC->TextOut( (int)new_posi_x, m_bottom-5, str_x, 3 );
            old_posi_x = new_posi_x;
        }
      }
    }
}

void CHistDlg::DrawAxisLine(CDC* pDC)
{
    //x 轴描画
    DrawAxisXLine(pDC);

    //y 轴描画
    DrawAxisYLine(pDC);

    //辅助线描画
    DrawSubLine(pDC);
}

//x 轴描画
void CHistDlg::DrawAxisXLine(CDC* pDC)
{
    int i;
    CString str_x;

    //线的幅宽        1
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 坐标
    //x 轴
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*
    m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);

    //小刻度
```



```
    for( i=0 ; i<(int)(m_max_value_x/m_small_x+1); i++ ){

      //x 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
      pDC->LineTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+2);
    }

    //大刻度
    for( i=0 ; i<(int)(m_max_value_x/m_large_x+1); i++ ){

      //x 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
      pDC->LineTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+4);
    }

 pDC->SelectObject(pOldPen);
}

//y 轴描画
void CHistDlg::DrawAxisYLine(CDC* pDC)
{
    int i;
    CString str_y;

//线的幅宽          1
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //y 坐标
    //y 轴左
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2, m_bottom-m_shift_val-
    (int)((float)m_max_value_y*m_zoom_y*m_inzoom_y));
```



```
    //小刻度
    for( i=0 ; i<(int)(m_max_value_y/m_small_y+1) ; i++ ){

      //y 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2 , m_bottom-m_shift_val-
      (int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));
      pDC->LineTo(m_left+m_shift_val*2-2, m_bottom-m_shift_val-
      (int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));
    }

    //大刻度
    for( i=0 ; i<(int)(m_max_value_y/m_large_y+1) ; i+- ){
      pDC->MoveTc(m_left+m_shift_val*2,

      //y 轴刻度
      m_bottom-m_shift_val-(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y));
      pDC->LineTo(m_left+m_shift_val*2-4,
      m_bottom-m_shift_val-(int)(m_large_ y*(float)i*m_zoom_y*m_inzoom_y));
    }

    pDC->SelectObject(pOldPen);
}

//辅助线描画
void CHistDlg::DrawSubLine(CDC* pDC)
{
    int st_x, st_y, end_x, end_y, i;

    //线的幅宽
    CPen  Pen(PS_DOT, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 轴
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*m_zoom_x*
    m_inzoom_x), m_bottom-m_shift_val);
    for( i=1 ; i<6 ; i++ ){
```



```
        st_x = m_left+m_shift_val*2;
        st_y = m_bottom-m_shift_val-(int)(m_large_y*(float)i*
        m_zoom_y*m_inzoom_y);
        end_x = m_left+m_shift_val*2+(int)(m_max_value_x*
        m_zoom_x*m_inzoom_x);
        end_y = m_bottom-m_shift_val-(int)(m_large_y*(float)i*
        m_zoom_y*m_inzoom_y);
        pDC->MoveTo( st_x,  st_y);
        pDC->LineTo(end_x, end_y);
    }
    pDC->SelectObject(pOldPen);
}

//直方图描画
void CHistDlg::DrawGrayValueMono( CDC* pDC )
{
    int i, st_x, st_gray;
    CPen  PenD(PS_SOLID, 1, RGB(255,0,0) );
    CPen* pOldPen = pDC->SelectObject(&PenD);

    UpdateData(TRUE);

    for( i=0 ; i<=255 ; i++ ){
       pDC->SelectObject(&PenD) ;
       st_x = m_left+m_shift_val*2-(int)((float)i*m_zoom_x*m_inzoom_x);
       st_gray = (int)(m_disp_array[i]*m_zoom_y*m_inzoom_y);
       pDC->MoveTo(st_x, m_bottom-m_shift_val) ;
       pDC->LineTo(st_x, m_bottom-m_shift_val - st_gray) ;
    }
    PenD.DeleteObject() ;
    pDC->SelectObject(pOldPen);
}

void CHistDlg::OnHistOriginal()
{
    // TODO: Add your control notification handler code here
```



```
    //表示原图像
    ::Disp_image(m_pImage_in);

    //计算直方图数列
    ::Histgram(m_pImage_in, m_xsize, m_ysize, m_hist);

    //图像表示更新
    m_pParent->Invalidate();

    //直方图的全体描画
    DispHistogram();

}

void CHistDlg::OnHistSmooth()
{
    // TODO: Add your control notification handler code here

    //平滑直方图数列
    ::Hist_smooth(m_hist, m_histSmooth);

    //直方图的全体描画
    DispHistogram();

}

void CHistDlg::OnHistAmplify()
{
    //亮度n倍
    ::Brightness_amplify(m_pImage_in, m_pImage_out, m_xsize, m_ysize, m_fn);

    //表示变换后图像
    ::Disp_image(m_pImage_out);

    //图像表示更新
    m_pParent->Invalidate();
```



```
    //计算直方图数列
    ::Histgram(m_pImage_out, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();
}

void CHistDlg::OnHistExpand()
{
    //亮度最大、最小值
int fmax, fmin;

    //计算原图像亮度的最大、最小值
    ::Brightness_range(m_pImage_in, m_xsize, m_ysize, &fmax, &fmin);

    //亮度范围扩张
    ::Brightness_expand(m_pImage_in,m_pImage_out,m_xsize,m_ysize,fmax, fmin);

    //表示变换后图像
    ::Disp_image(m_pImage_out);

    //图像表示更新
    m_pParent->Invalidate();

    //计算直方图数列
    ::Histgram(m_pImage_out, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();
}

void CHistDlg::OnHistPlan()
{
    //因为处理时间较长，请等候
    SetDlgItemText(IDC_HIST_INFO, "处理时间较长，请稍等候!!");
    //计算直方图数列
    ::Histgram(m_pImage_in, m_xsize, m_ysize, m_hist);
```



```
    //直方图平坦化
    ::Hist_plane(m_pImage_in, m_pImage_out, m_xsize, m_ysize, m_hist);

    //表示变换后图像
    ::Disp_image(m_pImage_out);

    //图像表示更新
    m_pParent->Invalidate();

    //计算直方图数列
    ::Histgram(m_pImage_out, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();

    SetDlgItemText(IDC_HIST_INFO, "");
}

void CHistDlg::OnHistImage()
{
   UpdateData(TRUE);

    //图像化直方图
    if(m_nHistType == 1)
    ::Hist_to_image(m_histSmooth, m_pImage_hist, m_xsize, m_ysize);
    else
    ::Hist_to_image(m_hist, m_pImage_hist, m_xsize, m_ysize);

    //表示直方图图像
    ::Disp_image(m_pImage_hist);
    //更新图像画面
    m_pParent->Invalidate();
}
```

附 1.3.2　直方图处理头文件

```
#if!defined(AFX_HISTDLG_H__F84E1158_A6B6_403A_8D68_851EA1DB30AE__INCLUDED_)
#define AFX_HISTDLG_H__F84E1158_A6B6_403A_8D68_851EA1DB30AE__INCLUDED_
```



```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// HistDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CHistDlg dialog

class CHistDlg : public CDialog
{
// Construction
public:
    CHistDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CHistDlg)
    enum { IDD = IDD_HIST DIALOG };
    int  m_nHistType;
    floatm_fn;
    //}}AFX_DATA

    //父亲窗口句柄
    CWnd  *m_pParent;

    //输入图像指针
    BYTE  *m_pImage_in;

    //输出图像指针
    BYTE  *m_pImage_out;

    //直方图图像指针
    BYTE  *m_pImage_hist;

    //图像宽度
    int    m_xsize;
```



```
    //图像高度
    int     m_ysize;

    //直方图的区域
//左
    int  m_left;
    //下
    int  m_bottom;
    //右
    int  m_right;
    //上
    int  m_top;

    //描画移动量
    int     m_shift_val;

    //x 轴刻度的最大值
    int  m_max_value_x;

    //y 轴刻度的最大值
    int  m_max_value_y;

    //x 轴文字刻度的变化量
    int  m_text_delta_x;

    //y 轴文字刻度的变化量
    int  m_text_delta_y;

    //x 轴小刻度的印记
    floatm_small_x;

    //x 轴大刻度的印记
    floatm_large_x;

    //y 轴小刻度的印记
    floatm_small_y;
```



```
//y 轴大刻度的印记
floatm_large_y;

//直方图配列
long    n_hist[256];

//平滑后直方图配列
long    m_histSmooth[256];

//百分比表示直方图配列
floatm_disp_array[256];

//最大百分比
floatm_max_percent;

//窗口领域宽度
float   m_resize_x;

//窗口领域高度
float   m_resize_y;

//x 方向扩大率
float   m_zoom_x;

//y 方向扩大率
float   m_zoom_y;

//内部扩大率，以基准高度的 n 倍表示
float   m_inzoom_y;
//以基准宽度的 n 倍表示
floatm_inzoom_x;

//轴上文字的表示、非表示
BOOL m_bAxisText;
BOOL    m_create;

//直方图的全体描画
```



```
    void DispHistogram();

    //参数、描画区域设定
    short DrawHistArea(CDC* pDC);

    //x、y轴文字的表示
    void DrawAxisText(CDC* pDC);

    //x、y轴描画
    void DrawAxisLine(CDC* pDC);

    //x轴描画
    void DrawAxisXLine(CDC* pDC);

    //y轴描画
    void DrawAxisYLine(CDC* pDC);

    //辅助线描画
    void DrawSubLine(CDC* pDC);

    //直方图描画
    void DrawGrayValueMono( CDC* pDC );

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHistDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void PostNcDestroy();
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CHistDlg)
    virtual BOOL OnInitDialog();
```

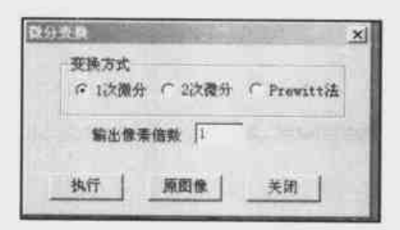


```
    afx_msg void OnPaint();
    afx_msg void OnHistOriginal();
    afx_msg void OnHistSmooth();
    afx_msg void OnHistAmplify();
    afx_msg void OnHistExpand();
    afx_msg void OnHistPlan();
    afx_msg void OnHistImage();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
//!defined(AFX_HISTDLG_H__F84E1158_A6B6_403A_8D68_851EA1DB30AE__INCLUDED_)
```

附 1.4 微分变换



附图 4

附 1.4.1 微分变换源程序

```
// DifferenDlg.cpp : implementation file
//
#include "stdafx.h"
#include "Imgc.h"
#include "DifferenDlg.h"
#include "Global.h"
#include "BaseList.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////
// CDifferenDlg dialog

CDifferenDlg::CDifferenDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CDifferenDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDifferenDlg)
    m_fRate = 1.0f;
    m_nModel = 0;
    //}}AFX_DATA_INIT

}
CDifferenDlg::~CDifferenDlg()
{


}

void CDifferenDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDifferenDlg)
    DDX_Text(pDX, IDC_DIFFERENTIAL_EDIT, m_fRate);
    DDV_MinMaxFloat(pDX, m_fRate, 1.f, 10.f);
    DDX_Radio(pDX, IDC_DIFFERENTIAL_RADIO, m_nModel);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDifferenDlg, CDialog)
    //{{AFX_MSG_MAP(CDifferenDlg)
    ON_BN_CLICKED(ID_DIFFERENTIAL_CLOSE, OnDifferentialClose)
    ON_BN_CLICKED(IDC_DIFFERENTIAL_ORIGINAL, OnDifferentialOriginal)
```



```
    ON_BN_CLICKED(ID_DIFFERENTIAL_EXECUTE, OnDifferentialExecute)
    ON_WM_CLOSE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDifferenDlg message handlers

BOOL CDifferenDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //获得图像大小
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
    m_pImageO = new BYTE[m_xsize*m_ysize];
    m_pImageD = new BYTE[m_xsize*m_ysize];

    //读入图像数据
    ReadImageData(m_pImageO);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CDifferenDlg::PostNcDestroy()
{
    // TODO: Add your specialized code here and/or call the base class

    //释放内存
    delete m_pImageO;
    delete m_pImageD;
```



```
    CDialog::PostNcDestroy();
}

BOOL CDifferenDlg::Create( CWnd* pParentWnd)
{
    //已构建时选择该窗口后退出
    if (m_hWnd != NULL) {
      SetFocus() ;
      return (TRUE) ;
    }

    //保存父亲窗口句柄
    m_pParentWnd = pParentWnd;

    //构建非模式窗口
    return CDialog::Create(IDD, pParentWnd);
}

void CDifferenDlg::OnDifferentialClose()
{
    // TODO: Add your control notification handler code here

    DestroyWindow();
}

void CDifferenDlg::OnDifferentialOriginal()
{
    // TODO: Add your control notification handler code here

    // 非灰度图像时退出
if(::GetImageType() != 8)
     return;

    //表示原图像
    ::Disp_image(m_pImageO);

    //更新图像画面
```



```
    m_pParentWnd->Invalidate();
}

void CDifferenDlg::OnDifferentialExecute()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    // 非灰度图像时退出
    if(::GetImageType() != 8)
    {
        AfxMessageBox("请读入灰度图像");
        return;
    }

    //1 次微分边缘检出
    if(m_nModel == 0)
      ::Differential(m_pImageO, m_pImageD, m_xsize, m_ysize, m_fRate);

    //2 次微分边缘检出
    else if(m_nModel == 1)
      ::Differential2(m_pImageO, m_pImageD, m_xsize, m_ysize, m_fRate);

    //Prewitt 法边缘检出
    else
      ::Prewitt(m_pImageO, m_pImageD, m_xsize, m_ysize, m_fRate);
    //表示处理结果
    ::Disp_image(m_pImageD);

    //更新图像画面
    m_pParentWnd->Invalidate();
}

void CDifferenDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default
```



```
    //解放非模式窗口
    DestroyWindow();
// CDialog::OnClose();

}
```

附 1.4.2 微分变换头文件

```
#if !defined(AFX_DIFFERENDLG_H__A02FAC1F_5724_4558_8C13_77A14DFA1B05__INCLUDED_)
#define
AFX_DIFFERENDLG_H__A02FAC1F_5724_4558_8C13_77A14DFA1B05__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DifferenDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDifferenDlg dialog

class CDifferenDlg : public CDialog
{
// Construction
public:
    CDifferenDlg(CWnd* pParent = NULL);   // standard constructor
    CDifferenDlg();
// Dialog Data
    //{{AFX_DATA(CDifferenDlg)
    enum { IDD = IDD_DIFFERENTIAL_DLG };
    floatm_fRate;
    int  m_nModel;
    //}}AFX_DATA

    //创建非模式窗口
    BOOL Create( CWnd* pParentWnd);

    //父亲窗口句柄
```



```
    CWnd *m_pParentWnd;

    //图像宽度
    int m_xsize;

    //图像高度
    int m_ysize;

    //原始图像数据指针
    BYTE *m_pImageO;

    //微分图像数据指针
    BYTE *m_pImageD;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDifferenDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void PostNcDestroy();
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CDifferenDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnDifferentialClose();
    afx_msg void OnDifferentialOriginal();
    afx_msg void OnDifferentialExecute();
    afx_msg void OnClose();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
```



```
// Microsoft Visual C++ will insert additional declarations immediately before
   the previous line.

#endif
//!defined(AFX_DIFFERENDLG_H__A02FAC1F_5724_4558_8C13_77A14DFA1B05__INCLUDED_)
```

附 1.5 特征提出



附图 5

附 1.5.1 特征提取源程序

```
// CAbstractDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "CAbstractDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
```



```
// CCAbstractDlg dialog
CCAbstractDlg::CCAbstractDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CCAbstractDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CCAbstractDlg)
   m_fRadioMax = 0.0f;
   m_fRadioMin = 0.0f;
   m_fRadioMax2 = 1.0f;
   m_fRadioMin2 = 0.1f;
   m_fAreaMax = 0.0f;
   m_fAreaMax2 = 100.0f;
   m_fAreaMin = 0.0f;
   m_fAreaMin2 = 1.0f;
   //}}AFX_DATA_INIT

   //保存表示画面句柄
   m_pParent = pParent;

   //初始化
   m_pimage_in = NULL;
   m_pimage_label = NULL;
   m_pimage_out = NULL;

   m_fsize = NULL;
   m_fradio = NULL;
   m_flength = NULL;

   m_ncenter_x = NULL;
   m_ncenter_y = NULL;
}

void CCAbstractDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCAbstractDlg)
    DDX_Text(pDX, IDC_ABSTRACT_RADIO_MAX, m_fRadioMax);
    DDX_Text(pDX, IDC_ABSTRACT_RADIO_MIN, m_fRadioMin);
```



```
    DDX Text(pDX, IDC_ABSTRACT_RADIO_MAX2, m_fRadioMax2);
    DDX_Text(pDX, IDC_ABSTRACT_RADIO_MIN2, m_fRadioMin2);
    DDX_Text(pDX, IDC_ABSTRACT_AERA_MAX, m_fAreaMax);
    DDX_Text(pDX, IDC_ABSTRACT_AREA_MAX2, m_fAreaMax2);
    DDX_Text(pDX, IDC_ABSTRACT_AREA_MIN, m_fAreaMin);
    DDX_Text(pDX, IDC_ABSTRACT_AREA_MIN2, m_fAreaMin2);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CCAbstractDlg, CDialog)
    //{{AFX_MSG_MAP(CCAbstractDlg)
    ON_BN_CLICKED(IDC_ABSTRACT_ORIGIN, OnAbstractOrigin)
    ON_BN_CLICKED(IDC_ABSTRACT_PARAMETER, OnAbstractParameter)
    ON_BN_CLICKED(IDC_ABSTRACT_AREA, OnAbstractArea)
    ON_BN_CLICKED(IDC_ABSTRACT_RATIO, OnAbstractRatio)
    ON_BN_CLICKED(IDC_ABSTRACT_RESULT, OnAbstractResult)
    ON_BN_CLICKED(IDC_ABSTRACT_DATA_VIEW, OnAbstractDataView)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCAbstractDlg message handlers

BOOL CCAbstractDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // TODO: Add extra initialization here

   //获取图像尺寸
   m_xsize = ::GetXSize();
   m_ysize = ::GetYSize();

   //分配内存
   m_pimage_in = new BYTE[m_xsize*m_ysize];
   m_pimage_label = new BYTE[m_xsize*m_ysize];
   m_pimage_out = new BYTE[m_xsize*m_ysize];
```



```
    //读入表示的图像数据
    ::ReadImageData(m_pimage_in);

    //区域个数初始化
    m_ncnt = 0;

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CCAbstractDlg::OnCancel()
{
    // TODO: Add extra cleanup here

    //释放内存

    if(m_pimage_in)
    {
      delete [] m_pimage_in;
      m_pimage_in = NULL;
    }
    if(m_pimage_label)
    {
      delete [] m_pimage_label;
      m_pimage_label = NULL;
    }
    if(m_pimage_out)
    {
      delete [] m_pimage_out;
      m_pimage_out = NULL;
    }

    if(m_fsize)
    {
      delete [] m_fsize;
      m_fsize = NULL;
```



```
    }
    if(m_fradio)
    {
      delete [] m_fradio;
      m_fradio = NULL;
    }

    if(m_flength)
    {
      delete [] m_flength;
      m_flength = NULL;
    }

    if(m_ncenter_x)
    {
      delete [] m_ncenter_x;
      m_ncenter_x = NULL;
    }

    if(m_ncenter_y)
    {
      delete [] m_ncenter_y;
      m ncenter y = NULL;
    }
    CDialog::OnCancel();
}


void CCAbstractDlg::OnAbstractOrigin()
{
    // TODO: Add your control notification handler code here

    //表示原图像
    ::Disp_image(m_pimage_in);

    //更新画面
    m_pParent->Invalidate();
```



```
}

void CCAbstractDlg::OnAbstractParameter()
{
    // TODO: Add your control notification handler code here

    int i, j;

    //消除周围像素
    for( i = 0; i < m_xsize; i++)
    {
      *(m_pimage_in + i) = 0;
      *(m_pimage_in + m_xsize + i) = 0;
      *(m_pimage_in + (m_ysize-1)*m_xsize + i) = 0;
      *(m_pimage_in + (m_ysize-2)*m_xsize + i) = 0;
    }
    for( j = 0; j < m_ysize; j++)
    {
      *(m_pimage_in + j*m_xsize ) = 0;
      *(m_pimage_in + j*m_xsize + 1 ) = 0;
      *(m_pimage_in + j*m_xsize + m_xsize-1) = 0;
      *(m_pimage_in + j*m_xsize + m_xsize-2) = 0;
    }
    //贴标签处理（区域处理）
    int nRnt=Labeling(m_pimage_in,m_pimage_label,m_xsize,m_ysize,&m_ncnt);

    //标签个数太多
    if(nRnt == -1)
    {
      m_ncnt = 0;
      return;
    }

    //没有测试对象
    if(m_ncnt == 0)
      return;
```



```
//分配内存
m_fradio = new float[m_ncnt];
m_fsize = new float[m_ncnt];
m_flength = new float[m_ncnt];
m_ncenter_x = new int[m_ncnt];
m_ncenter_y = new int[m_ncnt];

//测算特征数据
::Features(m_pimage_label,m_pimage_out,m_xsize,m_ysize,m_ncnt,m_fsize,
         m_flength, m_fradio, m_ncenter_x, m_ncenter_y);

//计算最大面积
m_fAreaMax = 0;
for(i=0; i<m_ncnt; i++)
m_fAreaMax = max(m_fAreaMax, m_fsize[i]);

//计算最小面积
m_fAreaMin = m_fAreaMax;
for(i=0; i<m_ncnt; i++)
  m_fAreaMin = min(m_fAreaMin, m_fsize[i]);

//计算最大圆形度
m_fRadioMax = 0;
for(i=0; i<m_ncnt; i++)
  m_fRadioMax = max(m_fRadioMax, m_fradio[i]);

//计算最小圆形度
m_fRadioMin = m_fRadioMax;
for(i=0; i<m_ncnt; i++)
  m_fRadioMin = min(m_fRadioMin, m_fradio[i]);

//更新窗口参数表示
UpdateData(FALSE);

//表示处理结果
::Disp_image(m_pimage_out);
```



```
    //更新画面
    m_pParent->Invalidate();

    //使数据表示键有效
    GetDlgItem(IDC_ABSTRACT_DATA_VIEW)->EnableWindow(TRUE);

}

void CCAbstractDlg::OnAbstractArea()
{
    // TODO: Add your control notification handler code here

    //获得窗口的参数
    UpdateData(TRUE);

    //没有测试对象
    if(m_ncnt <= 0) return;

    //抽出某面积范围的对象物
    ::Size_extract(m_pimage_label, m_pimage_out, m_xsize, m_ysize,
                  m_ncnt, m_fsize, m_fAreaMin2, m_fAreaMax2);

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();

    //使"结果图像"键有效
    GetDlgItem(IDC_ABSTRACT_RESULT)->EnableWindow(TRUE);

}

void CCAbstractDlg::OnAbstractRatio()
{
    // TODO: Add your control notification handler code here
```



```
    //获得窗口的参数
    UpdateData(TRUE);

    //没有处理对象时退出
    if(m_ncnt <= 0) return;

    //抽出具有某圆形度的对象物
    ::Ratio_extract(m_pimage_label, m_pimage_out, m_xsize, m_ysize,
    m_ncnt, m_fradio, m_fRadioMin2, m_fRadioMax2);

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();

    //使"结果图像"键有效
    GetDlgItem(IDC_ABSTRACT_RESULT)->EnableWindow(TRUE);

}

void CCAbstractDlg::OnAbstractResult()
{
    // TODO: Add your control notification handler code here

    //定义原始图像数据指针
    BYTE* image_ori;

    //定义处理结果图像指针
    BYTE* image_result;

    //定义保存图像名称的配列
    char cFileName[1024];

    //获得处理图像名称
    ::GetImageFileName(cFileName);
```



```
    //将图像名称变型
    CString sFileNane = cFileName;

    //读入原图像
    ::Load_original_image(sFileName);

    //分配内存
    image_ori = new BYTE[m_xsize*m_ysize];
    image_result = new BYTE[m_xsize*m_ysize];

    //读入原图像数据
    ::ReadImageData(image_ori);

    //在原图像上抽取
    ::Mask_copy(image_ori, image_result, m_pimage_out, m_xsize, m_ysize);

    //表示处理结果
    ::Disp_image(image_result);

    //更新画面
    n_pParent->Invalidate();

    //释放内存
    delete [] image_ori;
    delete [] image_result;

}

void CCAbstractDlg::OnAbstractDataView()
{
    // TODO: Add your control notification handler code here

    //代入测算数据
    int i;
    CString str1, str;
    str.Format("序号    面积    周长      圆形度      重心 \n");
```



```
    for(i=0; i<m_ncnt; i++)
    {
      str1.Format("%3d  %8d  %8.4f  %8.4f  (%d,%d)\n",i,(int)m_fsize[i],
      m_flength[i], m_fradio[i], m_ncenter_x[i], m_ncenter_y[i]);
      str = str + str1;
    }

    //表示测算数据
    AfxMessageBox(str);

}
```

附 1.5.2 特征提取头文件

```
#if !defined(AFX_CABSTRACTDLG_H__DCA89EC2_3F03_4A79_ADB1_AEA1AFFD334A__INCLUDED_)
#define
AFX_CABSTRACTDLG_H__DCA89EC2_3F03_4A79_ADB1_AEA1AFFD334A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CAbstractDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CCAbstractDlg dialog

class CCAbstractDlg : public CDialog
{
// Construction
public:
    CCAbstractDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CCAbstractDlg)
    enum { IDD = IDD_CHARACTER_ABSTRACT_DLG };
    floatm_fRadioMax;
    floatm_fRadioMin;
```



```
floatm_fRadioMax2;
floatm_fRadioMin2;
floatm_fAreaMax;
floatm_fAreaMax2;
floatm_fAreaMin;
floatm_fAreaMin2;
//}}AFX_DATA

//父亲窗口句柄
CWnd* m_pParent;

//输入图像指针
BYTE *m_pimage_in;

//输出图像指针
BYTE *m_pimage_out;

//区域标记图像指针
BYTE *m_pimage_label;

//面积数据指针
float *m_fsize;

//圆形度数据指针
float *m_fradio;

//长度数据指针
float *m_flength;

//重心坐标数据指针
int *m_ncenter_x;
int *m_ncenter_y;

//区域个数
int m_ncnt;

//图像宽度
```



```
    int m_xsize;

    //图像高度
    int m_ysize;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCAbstractDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CCAbstractDlg)
    virtual BOOL OnInitDialog();
    virtual void OnCancel();
    afx_msg void OnAbstractOrigin();
    afx_msg void OnAbstractParameter();
    afx_msg void OnAbstractArea();
    afx_msg void OnAbstractRatio();
    afx_msg void OnAbstractResult();
    afx_msg void OnAbstractDataView();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
//!defined(AFX_CABSTRACTDLG_H__DCA89EC2_3F03_4A79_ADB1_AEA1AFFD334A__INCLUDED_)
```



附 1.6　颜色变换



附图 6

附 1.6.1　颜色变换源程序

```
// ColorDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "ColorDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CColorDlg dialog

CColorDlg::CColorDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CColorDlg::IDD, pParent)
```



```
{
    //{{AFX_DATA_INIT(CColorDlg)
    m_fy = 1.0f;
    m_fsat = 1.0f;
    m_fhue = 0.0f;
    m_bR = 0;
    m_bG = 0;
    m_bB = 0;
    //}}AFX_DATA_INIT

    //保存表示画面句柄
    m_pParent = pParent;
}

void CColorDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CColorDlg)
    DDX_Text(pDX, IDC_COLOR_Y_PARAMETER, m_fy);
    DDX_Text(pDX, IDC_COLOR_SAT_PARAMETER, m_fsat);
    DDX_Text(pDX, IDC_COLOR_HUE_PARAMETER, m_fhue);
    DDX_Text(pDX, IDC_COLOR_BASE_R, m_bR);
    DDV_MinMaxByte(pDX, m_bR, 0, 255);
    DDX_Text(pDX, IDC_COLOR_BASE_G, m_bG);
    DDV_MinMaxByte(pDX, m_bG, 0, 255);
    DDX_Text(pDX, IDC_COLOR_BASE_B, m_bB);
    DDV_MinMaxByte(pDX, m_bB, 0, 255);
    //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CColorDlg, CDialog)
    //{{AFX_MSG_MAP(CColorDlg)
    ON_BN_CLICKED(IDC_COLOR_R, OnColorR)
    ON_BN_CLICKED(IDC_COLOR_G, OnColorG)
    ON_BN_CLICKED(IDC_COLOR_B, OnColorB)
    ON_BN_CLICKED(IDC_COLOR_RGB, OnColorRgb)
    ON_BN_CLICKED(IDC_COLOR_SAT_DISPLAY, OnColorSatDisplay)
    ON_BN_CLICKED(IDC_COLOR_HUE_DISPLAY, OnColorHueDisplay)
```



```
   ON_BN_CLICKED(IDC_COLOR_Y_DISPLAY, OnColorYDisplay)
   ON_BN_CLICKED(IDC_COLOR_RY_DISPLAY, OnColorRyDisplay)
   ON_BN_CLICKED(IDC_COLOR_BY_DISPLAY, OnColorByDisplay)
   ON_BN_CLICKED(IDC_COLOR_CHANG_IMAGE, OnColorChangImage)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CColorDlg message handlers

BOOL CColorDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   //获取图像尺寸
   m_xsize = ::GetXSize();
   m_ysize = ::GetYSize();

   //分配内存
   m_image_r = new BYTE[m_xsize*m_ysize];
   m_image_g = new BYTE[m_xsize*m_ysize];
   m_image_b = new BYTE[m_xsize*m_ysize];

   m_data_y = new int[m_xsize*m_ysize];
   m_data_ry = new int[m_xsize*m_ysize];
   m_data_by = new int[m_xsize*m_ysize];

   m_data_sat = new int[m_xsize*m_ysize];
   m_data_hue = new int[m_xsize*m_ysize];

   //读取图像数据
   ::ReadImageDataRGB(m_image_r, m_image_g, m_image_b);

   //获得读入图像的路径
   char cPath[256];
   ::GetImageFileName(cPath);
   m_sFilePath = cPath;
```



```
    //由 RGB 亮度色差变换
    ::Rgb_to_yc(m_image_r, m_image_g, m_image_b, m_data_y,
                m_data_ry, m_data_by, m_xsize, m_ysize);

    //由色差信号计算饱和度和色相
      ::C_to_SH(m_data_ry, m_data_by, m_data_sat, m_data_hue, m_xsize, m_ysize);

    return TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

void CColorDlg::OnCancel()
{
    //释放内存
    delete[] m_image_r;
    delete[] m_image_g;
    delete[] m_image_b;

    delete[] m_data_y;
    delete[] m_data_ry;
    delete[] m_data_by;

    delete[] m_data_sat;
    delete[] m_data_hue;

    CDialog::OnCancel();
}
void CColorDlg::OnColorR()
{

    //表示红色分量图像
    ::Disp_image(m_image_r);

    //更新画面
    m_pParent->Invalidate();
}
```



```
void CColorDlg::OnColorG()
{
    //表示绿色分量图像
    ::Disp_image(m_image_g);

    //更新画面
    m_pParent->Invalidate();
}

void CColorDlg::OnColorB()
{
    //表示蓝色分量图像
    ::Disp_image(m_image_b);

    //更新画面
    m_pParent->Invalidate();

}

void CColorDlg::OnColorRgb()
{
    //读入原图像
    ::Load_original_image(m_sFilePath);

    //读取图像数据
    ::ReadImageDataRGB(m_image_r, m_image_g, m_image_b);

    //由 RGB 亮度色差变换
    ::Rgb_to_yc(m_image_r, m_image_g, m_image_b, m_data_y,
            m_data_ry, m_data_by, m_xsize, m_ysize);

    //由色差信号计算饱和度和色相
    ::C_to_SH(m_data_ry,m_data_by,m_data_sat,m_data_hue,m_xsize,m_ysize);

    //更新画面
    m_pParent->Invalidate();
}
```



```
void CColorDlg::OnColorSatDisplay()
{
   BYTE *image;

   //分配内存
   image = new BYTE[m_xsize*m_ysize];

   //由饱和度数据变换灰度图像
   ::Sat_to_image(m_data_sat, image, m_xsize, m_ysize);

   //表示饱和度图像
   ::Disp_image(image);

   //更新画面
   m_pParent->Invalidate();

   //释放内存
   delete[] image;
}

void CColorDlg::OnColorHueDisplay()
{
   //获取窗口数据
   UpdateData(TRUE);
   double stdHue, sat;

//计算基准色色相  饱和度
   ::Rgb_to_SH(m_bR, m_bG, m_bB, &sat, &stdHue);

   //定义图像指针
   BYTE *image;

   //分配内存
   image = new BYTE[m_xsize*m_ysize];
   //由饱和度数据变换灰度图像
   ::Hue_to_image(m_data_sat, m_data_hue, stdHue, image, m_xsize, m_ysize);
```



```
    //表示饱和度图像
    ::Disp_image(image);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete[] image;
}

void CColorDlg::OnColorYDisplay()
{
    //定义图像指针
    BYTE *image;

    //分配内存
    image = new BYTE[m_xsize*m_ysize];

    //亮度信号图像化
    ::Yc_to_image(m_data_y, image, m_xsize, m_ysize);

    //表示灰度图像
    ::Disp_image(image);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete[] image;

}

void CColorDlg::OnColorRyDisplay()
{
    //定义图像指针
    BYTE *image;
```



```
    //分配内存
    image = new BYTE[m_xsize*m_ysize];

    //亮度信号图像化
    ::Yc_to_image(m_data_ry, image, m_xsize, m_ysize);

    //表示灰度图像
    ::Disp_image(image);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete[] image;
}

void CColorDlg::OnColorByDisplay()
{
    //定义图像指针
    BYTE *image;

    //分配内存
    image = new BYTE[m_xsize*m_ysize];

    //亮度信号图像化
    ::Yc_to_image(m_data_by, image, m_xsize, m_ysize);

    //表示灰度图像
    ::Disp_image(image);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete[] image;
}
```



```
void CColorDlg::OnColorChangImage()
{
    //定义亮度数据指针
    int *out_y;

    //定义饱和度数据指针
    int *out_sat;

    //定义色相数据指针
    int *out_hue;

    //定义 (r-y) 色差数据指针
    int *ry;

    //定义 (b-y) 色差数据指针
    int *by;

    //定义图像数据指针
    BYTE *image_r;
    BYTE *image_g;
    BYTE *image_b;

    //分配内存
    out_y = new int[m_xsize*m_ysize];
    out_sat = new int[m_xsize*m_ysize];
    out_hue = new int[m_xsize*m_ysize];

    ry = new int[m_xsize*m_ysize];
    by = new int[m_xsize*m_ysize];

    image_r = new BYTE[m_xsize*m_ysize];
    image_g = new BYTE[m_xsize*m_ysize];
    image_b = new BYTE[m_xsize*m_ysize];

    //获得窗口数据
    UpdateData(TRUE);
```



```
    //改变亮度、饱和度和色相
    ::Change_YSH(m_data_y, m_data_sat, m_data_hue, out_y,
       out_sat, out_hue, m_fy, m_fsat, m_fhue, m_xsize, m_ysize);

    //由饱和度和色相计算色差信号
    ::SH_to_C(out_sat, out_hue, ry, by, m_xsize, m_ysize);

    //由亮度 色差变换R、G、B信号
      ::Yc_to_rgb(out_y, ry, by, image_r, image_g,
            image_b, m_xsize, m_ysize);

    //表示变换后图像
    ::Disp_imageRGB( image r, image_g, image b);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete[] out_y;
    delete[] out_sat;
    delete[] out_hue;

    delete[] ry;
    delete[] by;
    delete[] image_r;
    delete[] image_g;
    delete[] image_b;
}
```

附 1.6.2 颜色变换头文件

```
#if!defined(AFX_COLORDLG_H__F1A327E2_4859_41ED_A076_752356A9F5D7__INCLUDED_)
#define AFX_COLORDLG_H__F1A327E2_4859_41ED_A076_752356A9F5D7__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ColorDlg.h : header file
```



```
//

///////////////////////////////////////////////////////////////////////////
// CColorDlg dialog

class CColorDlg : public CDialog
{
// Construction
public:
   CColorDlg(CWnd* pParent = NULL);   // standard constructor

   CWnd* m_pParent;
// Dialog Data
   //{{AFX_DATA(CColorDlg)
   enum { IDD = IDD_COLOR_DIALOG };
   float   m_fy;
   float   m_fsat;
   float   m_fhue;
   BYTE    m_bR;
   BYTE    m_bG;
   BYTE    m_bB;
   //}}AFX_DATA

   //图像R分量数据指针
   BYTE *m_image_r;

   //图像G分量数据指针
   BYTE *m_image_g;

   //图像B分量数据指针
   BYTE *m_image_b;

   //保存图像文件的路径
   CString m_sFilePath;

   //亮度数据指针
   int *m_data_y;
```



```
    //色差 (R-Y) 数据指针
    int *m_data_ry;

//色差 (B-Y) 数据指针
int *m_data_by;

    //饱和度数据指针
    int *m_data_sat;

    //色相数据指针
    int *m_data_hue;

    //图像宽度
    int m_xsize;

    //图像高度
    int m_ysize;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CColorDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CColorDlg)
    afx_msg void OnColorR();
    virtual BOOL OnInitDialog();
    virtual void OnCancel();
    afx_msg void OnColorG();
    afx_msg void OnColorB();
    afx_msg void OnColorRgb();
```

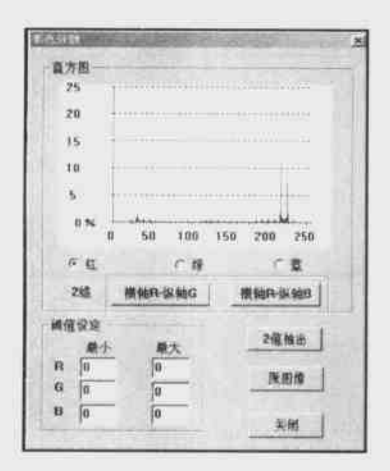


```
    afx_msg void OnColorSatDisplay();
    afx_msg void OnColorHueDisplay();
    afx_msg void OnColorYDisplay();
    afx_msg void OnColorRyDisplay();
    afx_msg void OnColorByDisplay();
    afx_msg void OnColorChangImage();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
    the previous line.

#endif
// !defined(AFX_COLORDLG_H__F1A327E2_4859_41ED_A076_752356A9F5D7__INCLUDED_)
```

附 1.7 彩色分割



附图 7

附 1.7.1 彩色分割源程序

```
// ColAbstractDlg.cpp : implementation file
//
```



```
#include "stdafx.h"
#include "Imgc.h"
#include "Global.h"
#include "BaseList.h"
#include "ColAbstractDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CColAbstractDlg dialog

CColAbstractDlg::CColAbstractDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CColAbstractDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CColAbstractDlg)
   m_nHistType = 0;
   m_nRmin = 0;
   m_nRmax = 0;
   m_nGmin = 0;
   m_nGmax = 0;
   m_nBmin = 0;
   m_nBmax = 0;
   //}}AFX_DATA_INIT

   //保存表示画面句柄
   m_pParent = pParent;
}

void CColAbstractDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CColAbstractDlg)
   DDX_Radio(pDX, IDC_COLOR_ABSTRACT_HIST_R, m_nHistType);
```



```
    DDX_Text(pDX, IDC_COLOR_R_MIN, m_nRmin);
    DDX_Text(pDX, IDC_COLOR_R_MAX, m_nRmax);
    DDX_Text(pDX, IDC_COLOR_G_MIN, m_nGmin);
    DDX_Text(pDX, IDC_COLOR_G_MAX, m_nGmax);
    DDX_Text(pDX, IDC_COLOR_B_MIN, m_nBmin);
    DDX_Text(pDX, IDC_COLOR_B_MAX, m_nBmax);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CColAbstractDlg, CDialog)
    //{{AFX_MSG_MAP(CColAbstractDlg)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_COLOR_ABSTRACT_HIST_R, OnColorAbstractHistR)
    ON_BN_CLICKED(IDC_COLOR_ABSTRACT_HIST_G, OnColorAbstractHistG)
    ON_BN_CLICKED(IDC_COLOR_ABSTRACT_B, OnColorAbstractB)
    ON_BN_CLICKED(IDC_COLOR_ABSTRACT_BINARY, OnColorAbstractBinary)
    ON_BN_CLICKED(IDC_COLOR_ABSTRACT_ORIGIN, OnColorAbstractOrigin)
    ON_BN_CLICKED(IDD_COLOR_HIST_IMAGE_RG, OnColorHistImageRg)
    ON_BN_CLICKED(IDD_COLOR_HIST_IMAGE_RB, OnColorHistImageRb)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CColAbstractDlg message handlers

BOOL CColAbstractDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   //描画移动量
   m_shift_val = 30;

   //左
   m_left = 0;

   //上
   m_top = 0;
```



```
    //右
    m_right = 200;

    //下
    m_bottom = 100;

    //轴的数据表示
    m_bAxisText = TRUE;

    //获得图像尺寸
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
    m_pImage_r = new BYTE[m_xsize*m_ysize];
    m_pImage_g = new BYTE[m_xsize*m_ysize];
    m_pImage_b = new BYTE[m_xsize*m_ysize];
    m_pImage_hist = new BYTE[m_xsize*m_ysize];

    //读入图像数据
    ReadImageDataRGB(m_pImage_r, m_pImage_g, m_pImage_b);

    //获得读入图像的路径
    char cPath[256];
    ::GetImageFileName(cPath);
    m_sFilePath = cPath;

    m_create = 1;
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
void CColAbstractDlg::PostNcDestroy()
{
    //解放内存
    delete[] m_pImage_r;
    delete[] m_pImage_g;
```



```
    delete[] m_pImage_b;
    delete[] m_pImage_hist;

    CDialog::PostNcDestroy();
}

void CColAbstractDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    //   获取绘制坐标的文本框
    CWnd* pWnd = GetDlgItem(IDC_COLOR_ABSTRACT_HIST);

    // 画面更新
    pWnd->Invalidate();
    pWnd->UpdateWindow();

    UpdateData(TRUE);

    //计算直方图数列
    if(m_nHistType == 0)
      ::Histgram(m_pImage_r, m_xsize, m_ysize, m_hist);
    else if(m_nHistType == 1)
      ::Histgram(m_pImage_g, m_xsize, m_ysize, m_hist);
    else
      ::Histgram(m_pImage_b, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();

    // Do not call CDialog::OnPaint() for painting messages
}
void CColAbstractDlg::DispHistogram()
{
    CDC* pDC;
    CRect rect;
    CWnd* pWnd = GetDlgItem(IDC_COLOR_ABSTRACT_HIST);
```



```
    pWnd->GetClientRect( &rect );

    //左
m_left = rect.left + 2;

//上
    m_top = rect.top + 8;

    //右
    m_right = rect.right - 2;

    //下
    m_bottom = rect.bottom - 2;

    //移动量
    m_shift_val = 30;

    //获得资源的关系
    pDC = pWnd->GetDC();

    //确定窗口参数
    UpdateData(TRUE);

    //计算直方图的百分比
    ::CalHistPercent(m_hist, m_disp_array, m_max_percent);

    //设定 y 轴最大值
    m_max_value_y = (int)(m_max_percent+1);

    //范围定为 0~100
    if( m_max_value_y >= 100 ) m_max_value_y = 100;
    if( m_max_value_y <= 0 )   m_max_value_y = 0;
    //参数、描画区域设定
    short ret = DrawHistArea(pDC);
    if( ret == OK ){

      //x、y 轴的描画
```



```
        DrawAxisLine(pDC);

        //x、y 轴文字的表示
        DrawAxisText(pDC);

        //亮度值的表示
        DrawGrayValueMono(pDC);
    }

    //资源关系的解放
    pWnd->ReleaseDC(pDC);

}

short CColAbstractDlg::DrawHistArea(CDC* pDC)
{
    int i, max_value, min_value;

    //x 轴
    //内部放大率
    //n 倍表示基准幅
    m_inzoom_x     =  (float)1;

//x 轴的小刻度
    m_small_x    =  (float)5;

//x 轴的大刻度
    m_large_x    =  (float)25;

//x 轴文字表示刻度的变化量
    m_text_delta_x  =  25;

    //x 轴的最大值
    m_max_value_x   =  255;

    //y 轴
    // 5, 10, 25, 50, 100
```



```
max_value = 100;
min_value = 100/2;

//求最大比例
for( i=0 ; i<5 ; i++ ){
   if( (min_value < m_max_value_y)&&( m_max_value_y <= max_value ) ){
      break;
   }
   else{
      max_value = min_value;
      if( min_value < 5 ){ min_value = 2; break; }
      if( min_value == 25 ) min_value = 10;
      else  min_value = min_value/2;
  }
}
m_max_value_y = max_value;

m_text_delta_y = m_max_value_y / 5;//1/5
if(m_text_delta_y == 0) m_text_delta_y = 1;
m_large_y = (float)m_text_delta_y; //==
m_small_y = m_large_y / (float)10;//1/10

//设定内部扩大率
if( m_max_value_y == 0 ) m_inzoom_y = (float)1;

//内部扩大率
else m_inzoom_y =  100 / (float)m_max_value_y ;

//
//右上领域的宽度、高度  m_resize_x, m_resize_y
m_resize_x = (float)(m_right - m_left);
m_resize_y = (float)(m_bottom - m_top);
//x 界限
if( m_resize_x < 200 ) return(NG);
if( m_shift_val < 30 ) return(NG);

//y 界限
```



```
    if( m_resize_y < 100 ) return(NG);

    //基准长x
m_zoom_x = ( m_resize_x - (float)60 ) / (float)280;

    //基准长y
    m_zoom_y =  m_resize_y  / (float) 120;

    //矩形区域描画
    CPen  Pen(PS_SOLID, 1, RGB(255,255,255) );
    CPen* pOldPen = pDC->SelectObject(&Pen);
    pDC->Rectangle( m_left, m_bottom+1,
                    m_left+(int)m_resize_x+1,
                    m_bottom-(int)m_resize_y );
    pDC->SelectObject(pOldPen);

    return(OK);
}

//x、y的刻度文字表示
void CColAbstractDlg::DrawAxisText( CDC* pDC )
{
    int i;
    float old_posi_y, new_posi_y;
    float old_posi_x, new_posi_x;

    CString str_x, str_y;

    pDC->SetTextAlign( TA_LEFT | TA_BOTTOM );
    pDC->SetTextColor( RGB(0,0,0) );

    //文字拥挤时拉开距离表示
    if( m_text_delta_y != 0 )
    {
      old_posi_y = (float)(m_bottom-20);

      //y轴的刻度文字表示
```



```
        str_y.Format("%3s", "0 %" );

    //"0%" 文字的微调整
        pDC->TextOut( m_left+17, (int)old_posi_y, str_y, 3 );

        for( i=1; i<(int)(m_max_value_y/m_text_delta_y+1) ; i++ ){
        new_posi_y = (float)m_bottom-(float)20-(m_text_delta_y*
        (float)i*m_zoom_y*m_inzoom_y);

            //y 轴数字间的宽度
            //m_text_interval_y=16
            if( (old_posi_y - (float)16) > new_posi_y ){

            //y 轴刻度数字
            str_y.Format("%3d", m_text_delta_y*i );
            pDC->TextOut( m_left+3, (int)new_posi_y, str_y, 3 );
            old_posi_y = new_posi_y;
          }
        }
      }


      if( m_text_delta_x != 0 ){
        old_posi_x = (float)(m_left+m_shift_val*2 - 15);

        //x 轴刻度数字
        str_x.Format("%3d", m_text_delta_x*0 );
        pDC->TextOut( (int)old_posi_x, m_bottom-5, str_x, 3 );

        for( i=1; i<(int)(m_max_value_x/m_text_delta_x+1) ; i++ ){
        new_posi_x = (float)m_left+(float)m_shift_val*2
        +(m_text_delta_x*(float)i*m_zoom_x*m_inzoom_x)-(float)15;
      //x 轴文字间隔
          if(  (new_posi_x - old_posi_x) > (float)28){

            //x 轴刻度数值
            str_x.Format("%3d", m_text_delta_x*i );
```



```
            pDC->TextOut( (int)new_posi_x, m_bottom-5, str_x, 3 );
            old_posi_x = new_posi_x;
        }
      }
    }
}

void CColAbstractDlg::DrawAxisLine(CDC* pDC)
{
    //x 轴描画
    DrawAxisXLine(pDC);

    //y 轴描画
    DrawAxisYLine(pDC);

    //辅助线描画
DrawSubLine(pDC);
}

//x 轴描画
void CColAbstractDlg::DrawAxisXLine(CDC* pDC)
{
    int i;
    CString str_x;

    //线的幅宽
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 坐标
    //x 轴
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*
    m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);

    //小刻度
    for( i=0 ; i<(int)(m_max_value_x/m_small_x+1); i++ ){
```



```
        //x 轴刻度
        pDC->MoveTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);

        pDC >LineTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+2);
    }

    //大刻度
    for( i=0 ; i<(int)(m_max_value_x/m_large_x+1); i++ ){

        //x 轴刻度
        pDC->MoveTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);

        pDC->LineTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+4);
    }

    pDC->SelectObject(pOldPen);
}

//y 轴描画
void CColAbstractDlg::DrawAxisYLine(CDC* pDC)
{
    int i;
    CString str_y;

    //线的幅宽          1
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);
    //y 坐标
    //y 轴左
    pDC->MoveTo(m_left+m_shift_val*2,
                m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2,
```



```
              m_bottom-m_shift_val-(int)((float)m_max_value_y*m_zoom_y*m_inzoom_y));

    //小刻度
    for( i=0 ; i<(int)(m_max_value_y/m_small_y+1) ; i++ ){

      //y 轴刻度
    pDC->MoveTo(m_left+m_shift_val*2 ,
m_bottom-m_shift_val-(int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));

    pDC->LineTo(m_left+m_shift_val*2-2,
m_bottom-m_shift_val-(int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));
    }

    //大刻度
    for( i=0 ; i<(int)(m_max_value_y/m_large_y+1) ; i++ ){

      //y 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2 ,
m_bottom-m_shift_val-(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y));

      pDC->LineTo(m_left+m_shift_val*2-4,
m_bottom-m_shift_val-(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y));
    }

    pDC->SelectObject(pOldPen);
}

//辅助线描画
void CColAbstractDlg::DrawSubLine(CDC* pDC)
{
    int st_x, st_y, end_x, end_y, i;

    //线的幅宽      1
    CPen  Pen(PS_DOT, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 轴
```



```
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*m_zoom_x*m_inzoom_x),
    m_bottom-m_shift_val);
    for( i=1 ; i<6 ; i++ ){
      st_x = m_left+m_shift_val*2;
      st_y = m_bottom-m_shift_val-
      (int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y);
      end_x = m_left+m_shift_val*2+
      (int)(m_max_value_x*m_zoom_x*m_inzoom_x);
      end_y = m_bottom-m_shift_val-
      (int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y);
      pDC->MoveTo( st_x,  st_y);
      pDC->LineTo(end_x, end_y);
    }
    pDC->SelectObject(pOldPen);
}

//直方图描画
void CColAbstractDlg::DrawGrayValueMono( CDC* pDC )
{
    UpdateData(TRUE);

    int i, st_x, st_gray;

    COLORREF color;

    if(m_nHistType == 0)
      color = RGB(255,0,0);
    if(m_nHistType == 1)
      color = RGB(0,255,0);
    if(m_nHistType == 2)
      color = RGB(0,0,255);
    CPen  PenD(PS_SOLID, 1, color );

    CPen* pOldPen = pDC->SelectObject(&PenD);

    UpdateData(TRUE);
```



```
    for( i=0 ; i<=255 ; i++ ){
      pDC->SelectObject(&PenD) ;
      st_x = m_left+m_shift_val*2+(int)((float)i*m_zoom_x*m_inzoom_x);
      st_gray = (int)(m_disp_array[i]*m_zoom_y*m_inzoom_y);
      pDC->MoveTo(st_x, m_bottom-m_shift_val) ;
      pDC->LineTo(st_x, m_bottom-m_shift_val - st_gray) ;
    }
    PenD.DeleteObject() ;
    pDC->SelectObject(pOldPen);
}

void CColAbstractDlg::OnColorAbstractHistR()
{
    UpdateData(TRUE);

    //计算直方图数列
    ::Histgram(m_pImage_r, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();
}

void CColAbstractDlg::OnColorAbstractHistG()
{
    //获得窗口数据
    UpdateData(TRUE);

    //计算直方图数列
    ::Histgram(m_pImage_g, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();
}

void CColAbstractDlg::OnColorAbstractB()
{
```



```
    //获得窗口数据
    UpdateData(TRUE);

    //计算直方图数列
    ::Histgram(m_pImage_b, m_xsize, m_ysize, m_hist);

    //直方图的全体描画
    DispHistogram();
}

void CColAbstractDlg::OnColorAbstractBinary()
{
    //获得窗口数据
    UpdateData(TRUE);

    //定义二值图像指针
    BYTE *image_out_r;
    BYTE *image_out_g;
    BYTE *image_out_b;

    //分配内存
    image_out_r = new BYTE[m_xsize*m_ysize];
    image_out_g = new BYTE[m_xsize*m_ysize];
    image_out_b = new BYTE[m_xsize*m_ysize];

    //R、G、B 的阈值处理
    ::Thresh_color(m_pImage_r, m_pImage_g, m_pImage_b,
    image_out_r, image_out_g, image_out_b,
    m_nRmin, m_nRmax, m_nGmin, m_nGmax, m_nBmin, m_nBmax,
    m_xsize, m_ysize);

    //表示彩色图像
    ::Disp_imageRGB(image_out_r, image_out_g, image_out_b);

    //更新画面
    m_pParent->Invalidate();
```



```
    //释放内存
    delete[] image_out_r;
    delete[] image_out_g;
    delete[] image_out_b;
}

void CColAbstractDlg::OnColorAbstractOrigin()
{
    //读入原图像
    ::Load_original_image(m_sFilePath);

    //读取图像数据
    ::ReadImageDataRGB(m_pImage_r, m_pImage_g, m_pImage_b);

    //更新画面
    m_pParent->Invalidate();
}

void CColAbstractDlg::OnColorHistImageRg()
{
    // TODO: Add your control notification handler code here
    ::Hist2_image(m_pImage_r, m_pImage_g, m_pImage_hist, m_xsize, m_ysize);

    //表示二维直方图像
    ::Disp_image(m_pImage_hist);

    //更新画面
    m_pParent->Invalidate();

}

void CColAbstractDlg::OnColorHistImageRb()
{
    // TODO: Add your control notification handler code here

//获得二维直方图
Hist2_image(m_pImage_r, m_pImage_b, m_pImage_hist, m_xsize, m_ysize);
```

```
    //表示二维直方图像
    ::Disp_image(m_pImage_hist);

    //更新画面
    m_pParent->Invalidate();
}
```

附 1.7.2 彩色分割头文件

```
#if!defined(AFX_COLABSTRACTDLG_H__8E6762C6_16EA_47B1_8408_EE3811B79C8_INCLUDED_)
#define AFX_COLABSTRACTDLG_H__8E6762C6_16EA_47B1_8408_EE3811B79C89__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ColAbstractDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CColAbstractDlg dialog

class CColAbstractDlg : public CDialog
{
// Construction
public:
    CColAbstractDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CColAbstractDlg)
    enum { IDD = IDD_COLOR_ABSTRACT_DIALOG };
    int    m_nHistType;
    int    m_nRmin;
    int    m_nRmax;
    int    m_nGmin;
    int    m_nGmax;
    int    m_nBmin;
    int    m_nBmax;
```



```
//}}AFX_DATA

//父亲窗口句柄
CWnd   *m_pParent;

//图像数据指针(R)
BYTE   *m_pImage_r;

//图像数据指针(G)
BYTE   *m_pImage_g;

//图像数据指针(B)
BYTE   *m_pImage_b;

//直方图图像指针
BYTE   *m_pImage_hist;

//保存图像文件路径
CString m_sFilePath;

//图像宽度
int     m_xsize;

//图像高度
int     m_ysize;

//直方图区域左侧
int    m_left;

//直方图区域下方
int    m_bottom;

//直方图区域右侧
int    m_right;

//直方图区域上方
int    m_top;
```



```
//描画移动量
int     m_shift_val;

//x 轴刻度的最大值
int    m_max_value_x;

//y 轴刻度的最大值
int    m_max_value_y;

//x 轴文字刻度的变化量
int    m_text_delta_x;

//y 轴文字刻度的变化量
int    m_text_delta_y;

//x 轴小刻度的印记
float m_small_x;

//y 轴小刻度的印记
float m_small_y;

//x 轴大刻度的印记
float m_large_x;

//y 轴大刻度的印记
float m_large_y;

//直方图配列
long    m_hist[256];

//比例表示用直方图配列
float m_disp_array[256];

//最大百分比
float m_max_percent;
```



```
//窗口领域宽度
float  m_resize_x;

//窗口领域高度
float  m_resize_y;

//x 轴扩大率
float  m_zoom_x;

//y 轴扩大率
float  m_zoom_y;

//y 轴内部扩大率，以基准高度的 n 倍表示
float  m_inzoom_y;

//y 轴内部扩大率，以基准高度的 n 倍表示
float   m_inzoom_x;

//轴上文字的表示、非表示
BOOL    m_bAxisText;

BOOL    m_create;

//直方图的全体描画
void DispHistogram();

//参数、描画区域设定
short DrawHistArea(CDC* pDC);

//x、y 轴文字的表示
void DrawAxisText(CDC* pDC);

//x、y 轴描画
void DrawAxisLine(CDC* pDC);

//x 轴描画
```



```
	void DrawAxisXLine(CDC* pDC);

	//y 轴描画
	void DrawAxisYLine(CDC* pDC);

	//辅助线描画
	void DrawSubLine(CDC* pDC);

	//直方图描画
	void DrawGrayValueMono( CDC* pDC );

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CColAbstractDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	virtual void PostNcDestroy();
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CColAbstractDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg void OnColorAbstractHistR();
	afx_msg void OnColorAbstractHistG();
	afx_msg void OnColorAbstractB();
	afx_msg void OnColorAbstractBinary();
	afx_msg void OnColorAbstractOrigin();
	afx_msg void OnColorHistImageRg();
	afx_msg void OnColorHistImageRb();
	//}}AFX_MSG
	DECLARE MESSAGE_MAP()
};
```



```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
// !defined(AFX_COLABSTRACTDLG_H__8E6762C6_16EA_47B1_8408_EE3811B79C89__INCLUDED_)
```

附 1.8 图像合成



附图 8

附 1.8.1 图像合成源程序

```
// SynthDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "SynthDlg.h"
#include "Global.h"
```



```
#include "BaseList.h"

#include "ImgcDoc.h"
#include "ImgcView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSynthDlg dialog


CSynthDlg::CSynthDlg(CWnd* pParent /*=NULL*/)
   :CDialog(CSynthDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CSynthDlg)
   m_DispImage = -1;
   m_nHThd = 10;
   m_nSThdMin = 10;
   m_nSThdMax = 200;
   //}}AFX_DATA_INIT

   //初始化
   m_image_fr = NULL;
   m_image_fg = NULL;
   m_image_fb = NULL;

   m_image_ar = NULL;
   m_image_ag = NULL;
   m_image_ab = NULL;

   m_image_keyh = NULL;
   m_image_keys = NULL;
```



```
    m_bfRead = FALSE;
    m_baRead = FALSE;

    //保存表示画面句柄
    m_pParent = pParent;
}
void CSynthDlg::DoDataExchange(CDataExchange* pDX)
{
    Dialog::DoDataExchange(pDX);
    /{{AFX_DATA_MAP(CSynthDlg)
    DX_Radio(pDX, IDC_SYNTH_DISPLAY_F, m_DispImage);
    DX_Text(pDX, IDC_SYNTH_H_THDH, m_nHThd);
    DX_Text(pDX, IDC_SYNTH_S_THDH_MIN, m_nSThdMin);
    DX_Text(pDX, IDC_SYNTH_S_THDH_MAX, m_nSThdMax);
    /}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSynthDlg, CDialog)
    /{{AFX_MSG_MAP(CSynthDlg)
    N_BN_CLICKED(IDC_COLOR_READ_FIRST, OnColorReadFirst)
    N_BN_CLICKED(IDC_COLOR_READ_AFTER, OnColorReadAfter)
    N_BN_CLICKED(IDC_SYNTH_DISPLAY_F, OnSynthDisplayF)
    N_BN_CLICKED(IDC_SYNTH_DISPLAY_A, OnSynthDisplayA)
    N_BN_CLICKED(IDC_SYNTH_KH, OnSynthKh)
    N_BN_CLICKED(IDC_SYNTH_SYNTHH, OnSynthSynthh)
    N_BN_CLICKED(IDC_SYNTH_KS, OnSynthKs)
    N_BN_CLICKED(IDC_SYNTH_SYNTHS, OnSynthSynths)
    N_BN_CLICKED(IDC_SYNTH_SSYNTHS, OnSynthSsynths)
    /}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////////
// CSynthDlg message handlers

BOOL CSynthDlg::OnInitDialog()
{
    Dialog::OnInitDialog();
```



```
   eturn TRUE;  // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}
void CSynthDlg::PostNcDestroy()
{
   //解放内存
   if(!m_image_fr)
   {
     delete[] m_image_fr;
     m_image_fr = NULL;
   }
   if(m_image_fg != NULL)
   {
   delete[] m_image_fg;
   m_image_fg = NULL;
   }
   if(m image fb != NULL)
   {
     delete[] m_image_fb;
     m_image_fb = NULL;
   }
   if(m_image_keyh != NULL)
   {
     delete[] m_image_keyh;
     m_image_keyh = NULL;
   }
   if(m_image_keys != NULL)
   {
     delete[] m_image_keys;
     m_image_keys = NULL;
   }

   CDialog::PostNcDestroy();
}

void CSynthDlg::OnColorReadFirst()
```



```
{
    //读入前景位图文件
    ::Load_imagefile_omp();

    //获得前景文件名
    char cPath[256];
    ::GetImageFileName(cPath);
    m_sFilePathf = cPath;

    //获得图像大小
    m_xsizef = ::GetXSize();
    m_ysizef = ::GetYSize();

    //设定图像窗口大小
    ((CImgcView*)m_pParent)->SetWindowSize(m_xsizef, m_ysizef);

    //更新画面
    m_pParent->Invalidate();

    //判断图像格式
    if( ::GetImageType() != 24)
    {
      AfxMessageBox("请读入彩色图像。");
      return;
    }

    //消除原始图像数据
    if(!m_image_fr)
    {
      delete[] m_image_fr;
      m_image_fr = NULL;
    }
    if(m_image_fg != NULL)
    {
      delete[] m_image_fg;
      m_image_fg = NULL;
    }
```



```
if(m_image_fb != NULL)
{
  delete[] m_image_fb;
  m_image_fb = NULL;
}
if(m_image_keyh != NULL)
{
  delete[] m_image_keyh;
  m_image_keyh = NULL;
}

//重新分配内存
m_image_fr = new BYTE[m_xsizef*m_ysizef];
m_image_fg = new BYTE[m_xsizef*m_ysizef];
m_image_fb = new BYTE[m_xsizef*m_ysizef];
m_image_keyh = new BYTE[m_xsizef*m_ysizef];

//读入新的图像数据
::ReadImageDataRGB(m_image_fr, m_image_fg, m_image_fb);

//变更表示
m_DispImage = 0;
UpdateData(FALSE);

// "前景图像" 有效
GetDlgItem(IDC_SYNTH_DISPLAY_F)->EnableWindow(TRUE);

m_bfRead = TRUE;

//检查背景图像是否读入
if(m_baRead)
{
  // "合成键" 有效
  GetDlgItem(IDC_SYNTH_KH)->EnableWindow(TRUE);
  GetDlgItem(IDC_SYNTH_KS)->EnableWindow(TRUE);
}
```



```
}
void CSynthDlg::OnColorReadAfter()
{
   //读入背景位图文件
   ::Load_imagefile_bmp();

   //获得背景文件名
   char cPath[256];
   ::GetImageFileName(cPath);
   m_sFilePatha = cPath;

   //获得图像大小
   m_xsizea = ::GetXSize();
   m_ysizea = ::GetYSize();

   //设定图像窗口大小
   ((CImgcView*)m_pParent)->SetWindowSize(m_xsizea, m_ysizea);

   //更新画面
   m_pParent->Invalidate();

   //判断图像格式
   if( ::GetImageType() != 24)
   {
     AfxMessageBox("请读入彩色图像。");
     return;
   }

   //消除原始图像数据
   if(!m_image_ar)
   {
     delete[] m_image_ar;
     m_image_ar = NULL;
   }
   if(m_image_ag != NULL)
   {
     delete[] m_image_ag;
```



```
    m_image_ag = NULL;
  }
  if(m_image_ab != NULL)
  {
    delete[] m_image_ab;
    m_image_ab = NULL;
  }
  if(m_image_keys != NULL)
  {
    delete[] m_image_keys;
    m_image_keys = NULL;
  }

  //重新分配内存
  m_image_ar = new BYTE[m_xsizea*m_ysizea];
  m_image_ag = new BYTE[m_xsizea*m_ysizea];
  m_image_ab = new BYTE[m_xsizea*m_ysizea];
  m_image_keys = new BYTE[m_xsizea*m_ysizea];

  //读入新图像数据
  ::ReadImageDataRGB(m_image_ar, m_image_ag, m_image_ab);

  //变更表示
  m_DispImage = 1;
  UpdateData(FALSE);

  // "背景图像" 有效
  GetDlgItem(IDC_SYNTH_DISPLAY_A)->EnableWindow(TRUE);

  m_baRead = TRUE;

  //检查前景图像是否读入
  if(m_bfRead)
  {
    // "合成键" 有效
    GetDlgItem(IDC_SYNTH_KH)->EnableWindow(TRUE);
    GetDlgItem(IDC_SYNTH_KS)->EnableWindow(TRUE);
```



```
    }
}

void CSynthDlg::OnSynthDisplayF()
{
    //读入前景位图文件
    ::Load_original_image(m_sFilePathf);

    //获得图像大小
    m_xsizef = ::GetXSize();
    m_ysizef = ::GetYSize();

    //设定图像窗口大小
    ((CImgcView*)m_pParent)->SetWindowSize(m_xsizef, m_ysizef);

    //更新画面
    m_pParent->Invalidate();

    //消除老图像数据
    if(!m_image_fr)
    {
      delete[] m_image_fr;
      m_image_fr = NULL;
    }
    if(m_image_fg != NULL)
    {
      delete[] m_image_fg;
      m_image_fg = NULL;
    }
    if(m_image_fb != NULL)
    {
      delete[] m_image_fb;
      m_image_fb = NULL;
    }

    //重新分配内存
    m_image_fr = new BYTE[m_xsizef*m_ysizef];
```



```
    m_image_fg = new BYTE[m_xsizef*m_ysizef];
    m_image_fb = new BYTE[m_xsizef*m_ysizef];

    //读入新图像数据
    ::ReadImageDataRGB(m_image_fr, m_image_fg, m_image_fb);
}

void CSynthDlg::OnSynthDisplayA()
{
    //读入前景位图文件
    ::Load_original_image(m_sFilePatha);

    //获得图像大小
    m_xsizea = ::GetXSize();
    m_ysizea = ::GetYSize();

    //设定图像窗口大小
    ((CImgcView*)m_pParent)->SetWindowSize(m_xsizea, m_ysizea);

    //更新画面
    m_pParent->Invalidate();

    //消除老图像数据
    if(!m_image_ar)
    {
      delete[] m_image_ar;
      m_image_ar = NULL;
    }
    if(m_image_ag != NULL)
    {
      delete[] m_image_ag;
      m_image_ag = NULL;
    }
    if(m_image_ab != NULL)
    {
      delete[] m_image_ab;
      m_image_ab = NULL;
```



```
    }

    //重新分配内存
    m_image_ar = new BYTE[m_xsizea*m_ysizea];
    m_image_ag = new BYTE[m_xsizea*m_ysizea];
    m_image_ab = new BYTE[m_xsizea*m_ysizea];

    //读入新图像数据
    ::ReadImageDataRGB(m_image_ar, m_image_ag, m_image_ab);

}

void CSynthDlg::OnSynthKh()
{
    //获得窗口数据
    UpdateData(TRUE);

    //检查图像大小是否相同
    if(m_xsizef != m_xsizea || m_ysizea != m_ysizea)
    {
      AfxMessageBox("请读入两幅大小相同的图像。");
      return;
    }

    //生成硬合成键
    ::Hard_key(m_image_fr, m_image_fg, m_image_fb,
      m_image_keyh, m_xsizef, m_ysizef, m_nHThd);

    //有效硬合成命令键
    GetDlgItem(IDC_SYNTH_SYNTHH)->EnableWindow(TRUE);

    //表示合成键图像
    ::Disp_image(m_image_keyh);

    //更新画面
    m_pParent->Invalidate();
}
```



```
void CSynthDlg::OnSynthSynthh()
{
  //定义输出图像数据指针
  BYTE *image_out_r;
  BYTE *image_out_g;
  BYTE *image_out_b;

  //检查两幅图像的大小是否相同
  if(m_xsizef != m_xsizea || m_ysizea != m_ysizea)
  {
    AfxMessageBox("请读入两幅大小相同的图像。");
    return;
  }

  //代入数据
  int xsize = m_xsizef;
  int ysize = m_ysizef;

  //分配内存
  image_out_r = new BYTE[xsize*ysize];
  image_out_g = new BYTE[xsize*ysize];
  image_out_b = new BYTE[xsize*ysize];

  // 图像硬合成
  ::Synth(m_image_fr,m_image_fg,m_image_fb,m_image_ar,m_image_ag, m_image_ab,
  image_out_r, image_out_g, image_out_b, m_image_keyh, xsize, ysize);

  //表示变换后图像
  ::Disp_imageRGB( image_out_r, image_out_g, image_out_b);

  //更新画面
  m_pParent->Invalidate();

  //释放内存
  delete [] image_out_r;
  delete [] image_out_g;
  delete [] image_out_b;
```



```
}

void CSynthDlg::OnSynthKs()
{
    //获得窗口数据
    UpdateData(TRUE);

    //检查两幅图像的大小是否相同
    if(m_xsizef != m_xsizea || m_ysizea != m_ysizea)
    {
      AfxMessageBox("请读入两幅大小相同的图像。");
      return;
    }

    //生成软合成键
    ::Soft_key(m_image_fr, m_image_fg, m_image_fb,
     m_image_keys, m_xsizef, m_ysizef, m_nSThdMax, m_nSThdMin);

    //有效软合成命令键
    GetDlgItem(IDC_SYNTH_SYNTHS)->EnableWindow(TRUE);
    GetDlgItem(IDC_SYNTH_SSYNTHS)->EnableWindow(TRUE);

    //表示合成键图像
    ::Disp_image(m_image_keys);

    //更新画面
    m_pParent->Invalidate();
}

void CSynthDlg::OnSynthSynths()
{
    //定义输出图像数据指针
    BYTE *image_out_r;
    BYTE *image_out_g;
    BYTE *image_out_b;

    //检查两幅图像的大小是否相同
```



```
if(m_xsizef != m_xsizea || m_ysizea != m_ysizea)
{
  AfxMessageBox("请读入两幅大小相同的图像。");
  return;
}
//代入数据
int xsize = m_xsizef;
int ysize = m_ysizef;

//分配内存
image_out_r = new BYTE[xsize*ysize];
image_out_g = new BYTE[xsize*ysize];
image_out_b = new BYTE[xsize*ysize];

// 图像硬合成
::Synth(m_image_fr,m_image_fg,m_image_fb,m_image_ar, m_image_ag,m_image_ab,
image_out_r, image_out_g, image_out_b, m_image_keys, xsize, ysize);

//表示变换后的图像
::Disp_imageRGB( image_out_r, image_out_g, image_out_b);

//更新画面
m_pParent->Invalidate();

//释放内存
delete [] image_out_r;
delete [] image_out_g;
delete [] image_out_b;
}

void CSynthDlg::OnSynthSsynths()
{
//定义输出图像数据指针
BYTE *image_out_r;
BYTE *image_out_g;
BYTE *image_out_b;
```



```
    //检查两幅图像的大小是否相同
    if(m_xsizef != m_xsizea || m_ysizea != m_ysizea)
    {
      AfxMessageBox("请读入两幅大小相同的图像。");
      return;
    }

    //代入数据
    int xsize = m_xsizef;
    int ysize = m_ysizef;

    //分配内存
    image_out_r = new BYTE[xsize*ysize];
    image_out_g = new BYTE[xsize*ysize];
    image_out_b = new BYTE[xsize*ysize];

    // 图像硬合成（消除境界线）
    ::S_synth(m_image_fr,m_image_fg,m_image_fb,m_image_ar,m_image_ag,m_image_ab,
      image_out_r, image_out_g, image_out_b, m_image_keys, xsize, ysize);

    //表示变换后的图像
    ::Disp_imageRGB( image_out_r, image_out_g, image_out_b);

    //更新画面
    m_pParent->Invalidate();

    //解放内存
    delete [] image_out_r;
    delete [] image_out_g;
    delete [] image_out_b;
}
```

附 1.8.2 图像合成头文件

```
#if !defined(AFX_SYNTHDLG_H__EB9326A5_15F6_4F0A_B099_5789685D44B3__INCLUDED_)
#define AFX_SYNTHDLG_H__EB9326A5_15F6_4F0A_B099_5789685D44B3__INCLUDED_
```



```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SynthDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSynthDlg dialog

class CSynthDlg : public CDialog
{
// Construction
public:
   CSynthDlg(CWnd* pParent = NULL);   // standard constructor

   //父亲窗口句柄
   CWnd* m_pParent;

   //前景图像 R 分量指针
   BYTE *m_image_fr;

   //前景图像 G 分量指针
   BYTE *m_image_fg;

   //前景图像 B 分量指针
   BYTE *m_image_fb;

   //背景图像 R 分量指针
   BYTE *m_image_ar;

   //背景图像 G 分量指针
   BYTE *m_image_ag;

   //背景图像 B 分量指针
   BYTE *m_image_ab;

   //硬合成键数据指针
```



```
    BYTE *m_image_keyh;
    //软合成键数据指针
    BYTE *m_image_keys;

    //前景图像宽度
    int m_xsizef;

    //前景图像高度
    int m_ysizef;

    //背景图像宽度
    int m_xsizea;

    //背景图像高度
    int m_ysizea;

    //记录前景图像是否读入
    BOOL m_bfRead;

    //记录背景图像是否读入
    BOOL m_baRead;

    //前景图像文件路径
    CString m_sFilePathf;

    //背景图像文件路径
    CString m_sFilePatha;

// Dialog Data
    //{{AFX_DATA(CSynthDlg)
    enum { IDD = IDD_COLOR_SYNTH_DIALOG };
    int    m_DispImage;
    int    m_nHThd;
    int    m_nSThdMin;
    int    m_nSThdMax;
    //}}AFX_DATA
```



```
// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CSynthDlg)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   virtual void PostNcDestroy();
   //}}AFX_VIRTUAL

// Implementation
protected:

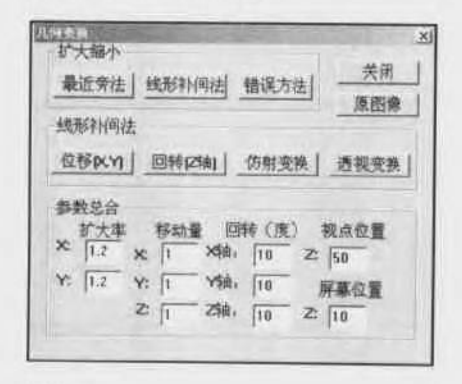
   // Generated message map functions
   //{{AFX_MSG(CSynthDlg)
   afx_msg void OnColorReadFirst();
   virtual BOOL OnInitDialog();
   afx_msg void OnColorReadAfter();
   afx_msg void OnSynthDisplayF();
   afx_msg void OnSynthDisplayA();
   afx_msg void OnSynthDisplayKh();
   afx_msg void OnSynthDisplayKs();
   afx_msg void OnSynthDisplaySynth();
   afx_msg void OnSynthDisplaySsynth();
   afx_msg void OnSynthKh();
   afx_msg void OnSynthSynthh();
   afx_msg void OnSynthKs();
   afx_msg void OnSynthSynths();
   afx_msg void OnSynthSsynths();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
//Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_SYNTHDLG_H__EB9326A5_15F6_4F0A_B099_5789685D44B3__INCLUDED_)
```



附 1.9　几何变换



附图 9

附 1.9.1　几何变换源程序

```
// DistortionDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "DistortionDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDistortionDlg dialog

CDistortionDlg::CDistortionDlg(CWnd* pParent /*=NULL*/)
```



```
    : CDialog(CDistortionDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDistortionDlg)
    m_fzx = 1.2f;
    m_fzy = 1.2f;
    m_fsx = 1.0f;
    m_fsy = 1.0f;
    m_fsz = 1.0f;
    m_ftx = 10.0f;
    m_fty = 10.0f;
    m_ftz = 10.0f;
    m_fpv = 50.0f;
    m_fps = 10.0f;
    //}}AFX_DATA_INIT

    //保存表示画面的句柄
    m_pParent = pParent;
}


void CDistortionDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDistortionDlg)
    DDX_Text(pDX, IDC_DISTORTION_RATE_X, m_fzx);
    DDX_Text(pDX, IDC_DISTORTION_RATE_Y, m_fzy);
    DDX_Text(pDX, IDC_DISTORTION_SHIFT_X, m_fsx);
    DDX_Text(pDX, IDC_DISTORTION_SHIFT_Y, m_fsy);
    DDX_Text(pDX, IDC_DISTORTION_SHIFT_Z, m_fsz);
    DDX_Text(pDX, IDC_DISTORTION_TURN_X, m_ftx);
    DDX_Text(pDX, IDC_DISTORTION_TURN_Y, m_fty);
    DDX_Text(pDX, IDC_DISTORTION_TURN_Z, m_ftz);
    DDX_Text(pDX, IDC_DISTORTION_POSITION_V, m_fpv);
    DDX_Text(pDX, IDC_DISTORTION_POSITION_S, m_fps);
    //}}AFX_DATA_MAP
}
```



```
BEGIN_MESSAGE_MAP(CDistortionDlg, CDialog)
   //{{AFX_MSG_MAP(CDistortionDlg)
   ON_BN_CLICKED(IDC_DISTORTION_SCALE_NEAR, OnDistortionScaleNear)
   ON_BN_CLICKED(IDC_DISTORTION_ORIGIN, OnDistortionOrigin)
   ON_BN_CLICKED(IDC_DISTORTION_SCALE, OnDistortionScale)
   ON_BN_CLICKED(IDC_DISTORTION_SHIFT, OnDistortionShift)
   ON_BN_CLICKED(IDC_DISTORTION_ROTATION, OnDistortionRotation)
   ON_BN_CLICKED(IDC_DISTORTION_AFFINE, OnDistortionAffine)
   ON_BN_CLICKED(IDC_DISTORTION_PERSPECTIVE, OnDistortionPerspective)
   ON_BN_CLICKED(IDC_DISTORTION_SCALE_ERROR, OnDistortionScaleError)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDistortionDlg message handlers

BOOL CDistortionDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   //获得图像大小
   m_xsize = ::GetXSize();
   m_ysize = ::GetYSize();

   //分配内存
   m_image_in = new BYTE[m_xsize*m_ysize];
   m_image_out = new BYTE[m_xsize*m_ysize];

   //读入图像数据
   ::ReadImageData(m_image_in);

   //获得读入图像的路径
   char cPath[256];
   ::GetImageFileName(cPath);
   m_sFilePath = cPath;

   return TRUE;  // return TRUE unless you set the focus to a control
```



```
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CDistortionDlg::PostNcDestroy()
{
    // 释放内存
    delete [] m_image_in;
    delete [] m_image_out;

    CDialog::PostNcDestroy();
}

void CDistortionDlg::OnDistortionScaleNear()
{
    //获得窗口参数
    UpdateData(TRUE);

    //扩大、缩小
    ::Scale_near(m_image_in, m_image_out, m_xsize, m_ysize, m_fzx, m_fzy);

    //表示处理结果
    ::Disp_image(m_image_out);

    //更新画面
    m_pParent->Invalidate();
}

void CDistortionDlg::OnDistortionScale()
{
    //获得窗口参数
    UpdateData(TRUE);

    //扩大、缩小
    ::Scale(m_image_in, m_image_out, m_xsize, m_ysize, m_fzx, m_fzy);

    //表示处理结果
    ::Disp_image(m_image_out);
```



```
    //更新画面
    m_pParent->Invalidate();
}
void CDistortionDlg::OnDistortionScaleError()
{
    //获得窗口参数
    UpdateData(TRUE);

    //扩大、缩小（错误方法）
    ::Scale_NG(m_image_in, m_image_out, m_xsize, m_ysize, m_fzx, m_fzy);

    //表示处理结果
    ::Disp_image(m_image_out);

    //更新画面
    m_pParent->Invalidate();

}

void CDistortionDlg::OnDistortionOrigin()
{
    //读入原图像
    ::Load_original_image(m_sFilePath);

    //读取图像数据
    ::ReadImageData(m_image_in);

    //更新画面
    m_pParent->Invalidate();
}

void CDistortionDlg::OnDistortionShift()
{
    //获得窗口参数
    UpdateData(TRUE);
```



```
    //位移
    ::Shift(m_image_in, m_image_out, m_xsize, m_ysize, m_fsx, m_fsy);

    //表示处理结果
    ::Disp_image(m_image_out);

    //更新画面
    m_pParent->Invalidate();
}

void CDistortionDlg::OnDistortionRotation()
{
    //获得窗口参数
    UpdateData(TRUE);

    //回转
    ::Rotation(m_image_in, m_image_out, m_xsize, m_ysize, m_ftz);

    //表示处理结果
    ::Disp_image(m_image_out);

    //更新画面
    m_pParent->Invalidate();
}

void CDistortionDlg::OnDistortionAffine()
{
    //获得窗口参数
    UpdateData(TRUE);

    //仿射变换 (移动、旋转、放大缩小)
    ::Affine(m_image_in, m_image_out, m_xsize, m_ysize, m_ftz,
      m_fzx, m_fzy, m_fsx, m_fsy);

    //表示处理结果
    ::Disp_image(m_image_out);
```



```
    //更新画面
    m_pParent->Invalidate();
}

void CDistortionDlg::OnDistortionPerspective()
{
    //获得窗口参数
    UpdateData(TRUE);

    //透视变换
    ::Perspective(m_image_in, m_image_out, m_xsize, m_ysize, m_fzx, m_fzy,
    m_fsx, m_fsy, m_fsz, m_ftz, m_ftx, m_fty, m_fpv, m_fps);

    //表示处理结果
    ::Disp_image(m_image_out);

    //更新画面
    m_pParent->Invalidate();
}
```

附 1.9.2　几何变换头文件

```
#if !defined(AFX_DISTORTIONDLG_H__B8F58936_2DF1_4226_99EA_AF40327BC1D5__INCLUDED_)
#define
AFX_DISTORTIONDLG_H__B8F58936_2DF1_4226_99EA_AF40327BC1D5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DistortionDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDistortionDlg dialog

class CDistortionDlg : public CDialog
{
// Construction
```

```
public:
	CDistortionDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CDistortionDlg)
	enum { IDD = IDD_DISTORTION_DIALOG };
	float m_fzx;
	float m_fzy;
	float m_fsx;
	float m_fsy;
	float m_fsz;
	float m_ftx;
	float m_fty;
	float m_ftz;
	float m_fpv;
	float m_fps;
	//}}AFX_DATA

	//父亲窗口句柄
	CWnd* m_pParent;

	//输入图像数据指针
	BYTE *m_image_in;

	//输入图像数据指针
	BYTE *m_image_out;

	//图像宽度
	int m_xsize;

	//图像高度
	int m_ysize;

	//图像文件路径
	CString m_sFilePath;

// Overrides
```



```
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDistortionDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	virtual void PostNcDestroy();
	//}}AFX_VIRTUAL

// Implementation
protected:

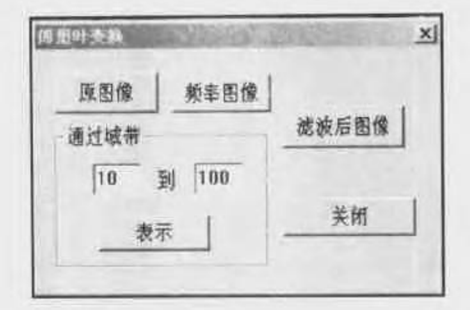
	// Generated message map functions
	//{{AFX_MSG(CDistortionDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnDistortionScaleNear();
	afx_msg void OnDistortionOrigin();
	afx_msg void OnDistortionScale();
	afx_msg void OnDistortionShift();
	afx_msg void OnDistortionRotation();
	afx_msg void OnDistortionAffine();
	afx_msg void OnDistortionPerspective();
	afx_msg void OnDistortionScaleError();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
// !defined(AFX_DISTORTIONDLG_H__B8F58936_2DF1_4226_99EA_AF40327BC1D5__INCLUDED_)
```



附 1.10 傅立叶变换



附图 10

附 1.10.1 傅立叶变换源程序

```
// FftDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "FftDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFftDlg dialog

CFftDlg::CFftDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CFftDlg::IDD, pParent)
{
```



```
    //{{AFX_DATA_INIT(CFftDlg)
    m_nFrom = 10;
    m_nTo = 100;
    //}}AFX_DATA_INIT

    //保存表示画面句柄
    m_pParent = pParent;
}

void CFftDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFftDlg)
    DDX_Text(pDX, IDC_FFT_FROM, m_nFrom);
    DDX_Text(pDX, IDC_FFT_TO, m_nTo);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CFftDlg, CDialog)
    //{{AFX_MSG_MAP(CFftDlg)
    ON_BN_CLICKED(IDC_FFT_ORIGIN, OnFftOrigin)
    ON_BN_CLICKED(IDC_FFT_IMAGE, OnFftImage)
    ON_BN_CLICKED(IDC_FFT_FILTER_IMAGE, OnFftFilterImage)
    ON_BN_CLICKED(IDC_FFT_DISPLAY, OnFftDisplay)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFftDlg message handlers

BOOL CFftDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    //获取图像尺寸
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();
```



```
    //分配内存
    m_image = new BYTE[m_xsize*m_ysize];

    m_image_fft = new BYTE[m_xsize*m_ysize];

    //读入表示的图像数据
    ::ReadImageData(m_image);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CFftDlg::PostNcDestroy()
{
    // TODO: Add your specialized code here and/or call the base class

    //释放内存
    delete[] m_image;
    delete[] m_image_fft;

    CDialog::PostNcDestroy();
}

void CFftDlg::OnFftOrigin()
{
    //表示原图像
    ::Disp_image(m_image);

    //更新画面
    m_pParent->Invalidate();
}

void CFftDlg::OnFftImage()
{
    //将 2 次 FFT 的变换结果图像化
    ::FFTImage(m_image, m_image_fft, m_xsize, m_ysize);
```



```
    //表示 FFT 图像
    ::Disp_image(m_image_fft);

    //更新画面
    m_pParent->Invalidate();
}

void CFftDlg::OnFftFilterImage()
{
    //获取窗口数据
    UpdateData(TRUE);

    //定义输出图像数据指针
    BYTE *image_out;
    //分配内存
    image_out = new BYTE[m_xsize*m_ysize];

    //图像的 FFT 变换、滤波处理、逆傅立叶变换
    ::FFTFilterImage(m_image, image_out, m_xsize, m_ysize, m_nFrom, m_nTo);

    //表示滤波后图像
    ::Disp_image(image_out);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete [] image_out;
}

void CFftDlg::OnFftDisplay()
{
    //获得窗口数据
    UpdateData(TRUE);

    //定义输出图像数据指针
```



```
    BYTE *image_out;

    //分配内存
    image_out = new BYTE[m_xsize*m_ysize];

    //图像的FFT变换及滤波处理
    ::FFTFilter(m_image, image_out, m_xsize, m_ysize, m_nFrom, m_nTo);

    //表示滤波后的频率图像
    ::Disp_image(image_out);

    //更新画面
    m_pParent->Invalidate();

    //释放内存
    delete [] image_out;
}
```

附 1.10.2 傅立叶变换头文件

```
#if !defined(AFX_FFTDLG_H__D0DF5C70_8D52_4818_AE44_997D49389B87__INCLUDED_)
#define AFX_FFTDLG_H__D0DF5C70_8D52_4818_AE44_997D49389B87__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FftDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFftDlg dialog

class CFftDlg : public CDialog
{
// Construction
public:
    CFftDlg(CWnd* pParent = NULL);   // standard constructor
```



```
// Dialog Data
    //{{AFX_DATA(CFftDlg)
    enum { IDD = IDD_FFT_DIALOG };
    int    m_nFrom;
    int    m_nTo;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFftDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual void PostNcDestroy();
    //}}AFX_VIRTUAL

    //父亲窗口句柄
    CWnd* m_pParent;

    //图像数据指针
    BYTE *m_image;

    //频率数据指针
    BYTE *m_image_fft;

    //图像宽度
    int m_xsize;
    //图像高度
    int m_ysize;

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CFftDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnFftOrigin();
    afx_msg void OnFftImage();
```

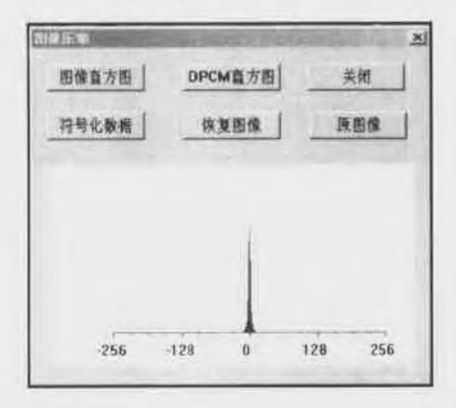


```
    afx_msg void OnFftFilterImage();
    afx_msg void OnFftDisplay();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
//!defined(AFX_FFTDLG_H__D0DF5C70_8D52_4818_AE44_997D49389B87__INCLUDED_)
```

附 1.11　图像压缩



附图 11

附 1.11.1　图像压缩源程序

```
// DpcmDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "DpcmDlg.h"
```



```
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDpcmDlg dialog

CDpcmDlg::CDpcmDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CDpcmDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDpcmDlg)
   // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT

   m_bDpcmHist = FALSE;

   //保存父亲窗口句柄
   m_pParent = pParent;
}

void CDpcmDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CDpcmDlg)
     // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDpcmDlg, CDialog)
   //{{AFX_MSG_MAP(CDpcmDlg)
   ON_BN_CLICKED(IDC_DPCM_HIST, OnDpcmHist)
   ON_WM_PAINT()
```



```
    ON_BN_CLICKED(IDC_DPCM_IMAGE_HIST, OnDpcmImageHist)
    ON_BN_CLICKED(IDC_DPCM_VLC, OnDpcmVlc)
    ON_BN_CLICKED(IDC_DPCM_VLC_IMAGE, OnDpcmVlcImage)
    ON_BN_CLICKED(IDC_DPCM_ORIGIN, OnDpcmOrigin)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDpcmDlg message handlers

BOOL CDpcmDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    //描画移动量
    m_shift_val = 30;

    //直方图区域左侧
    m_left = 0;

    //直方图区域上方
    m_top = 0;

    //直方图区域右侧
    m_right = 200;

    //直方图区域下方
    m_bottom = 100;

    //轴的数据表示
    m_bAxisText = TRUE;

    //获得图像尺寸
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
```



```
    m_pImage_in = new BYTE[m_xsize*m_ysize];
    m_pImage_out = new BYTE[m_xsize*m_ysize];
    m_pImage_buf = new BYTE[m_xsize*m_ysize];
    m_pImage_hist = new BYTE[m_xsize*m_ysize];

    //读入图像数据
    ::ReadImageData(m_pImage_in);

    //计算直方图数列
    ::Histgram(m_pImage_in, m_xsize, m_ysize, m_hist);

    //计算 DPCM 直方图数列
    ::Histgram_dpcm(m_pImage_in, m_xsize, m_ysize, m_hist_dpcm);

    m_create = 1;

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CDpcmDlg::PostNcDestroy()
{
    // TODO: Add your specialized code here and/or call the base class

    //解放内存
    delete[] m_pImage_in;
    delete[] m_pImage_out;
    delete[] m_pImage_buf;
    delete[] m_pImage_hist;

    CDialog::PostNcDestroy();
}

void CDpcmDlg::OnDpcmHist()
{
    //表示 DPCM 数据直方图
    m_bDpcmHist = TRUE;
```



```
    //直方图的全体描画
    DispHistogram();
}

void CDpcmDlg::OnDpcmImageHist()
{
    //表示图像数据直方图
    m_bDpcmHist = FALSE;

    //直方图的全体描画
    DispHistogram();
}

void CDpcmDlg::OnPaint()
{
    CPaintDC dc(this); // device context for painting

     // 获取绘制坐标的文本框
    CWnd* pWnd = GetDlgItem(IDC_DPCM_HIST_ARER);

    // 画面更新
    pWnd->Invalidate();
    pWnd->UpdateWindow();

    //获得窗口数据
    UpdateData(TRUE);

    //直方图的全体描画
    DispHistogram();

    // Do not call CDialog::OnPaint() for painting messages
}

void CDpcmDlg::DispHistogram()
{
    CDC* pDC;
```



```
CRect rect;
CWnd* pWnd = GetDlgItem(IDC_DPCM_HIST_ARER);
pWnd->GetClientRect( &rect );

//直方图区域左侧
m_left = rect.left + 2;

//直方图区域上方
m_top = rect.top + 8;

//直方图区域右侧
m_right = rect.right - 2;
//直方图区域下方
m_bottom = rect.bottom - 2;

//移动量
m_shift_val = 30;

//获得资源的关系
pDC = pWnd->GetDC();

//确定窗口参数
UpdateData(TRUE);

if( m_bDpcmHist )
  //计算 DPCM 数据百分比
  ::CalHistPercent_dpcm(m_hist_dpcm, m_disp_array_dpcm,
m_max_percent);
else
//计算原图像数据百分比
::CalHistPercent(m_hist, m_disp_array, m_max_percent);

//设定 Y 轴最大值
m_max_value_y = (int)(m_max_percent+1);

//范围定为 0~100
if( m_max_value_y >= 100 ) m_max_value_y = 100;
```



```
    if( m_max_value_y <= 0 )   m_max_value_y = 0;

    //参数、描画区域设定
    short ret = DrawHistArea(pDC);
    if( ret == OK ){
      //x、y 轴的描画
      DrawAxisLine(pDC);

      //x、y 轴文字的表示
      DrawAxisText(pDC);

      //亮度值的表示
      DrawGrayValueMono(pDC);
    }

    //释放资源关系
    pWnd->ReleaseDC(pDC);

}

short CDpcmDlg::DrawHistArea(CDC* pDC)
{
    int i, max_value, min_value;

    //x 轴
    if(m_bDpcmHist)
    {
      //DCPM 直方图

      //内部放大率, n 倍表示基准幅
      m_inzoom_x=(float)0.5;

      //x 轴的小刻度
      m_small_x=(float)32;

      //x 轴的大刻度
      m_large_x=(float)128;
```



```
        //x 轴文字表示刻度的变化量
        m_text_delta_x=128;

        //x 轴的最大值
        m_max_value_x=512;
    }
    else
    {
        //图像直方图

        //内部放大率，n 倍表示基准幅
        m_inzoom_x=(float)1;

        //x 轴的小刻度
        m_small_x=(float)5;

        //x 轴的大刻度
        m_large_x=(float)25;

        //x 轴文字表示刻度的变化量
        m_text_delta_x=25;

        //x 轴的最大值
        m_max_value_x=255;
    }

    //y 轴
    // 5, 10, 25, 50, 100
    max_value = 100;
    min_value = 100/2;

    //求最大比例
    for( i=0 ; i<5 ; i++ ){
        if( (min_value < m_max_value_y)&&( m_max_value_y <= max_value ) ){
            break;
        }
```

```
    else{
      max_value = min_value;
      if( min_value < 5 ){ min_value = 2; break; }
      if( min_value == 25 ) min_value = 10;
      else           min_value = min_value/2;
    }
  }
m_max_value_y = max_value;

m_text_delta_y = m_max_value_y / 5;//1/5
if(m_text_delta_y == 0) m_text_delta_y = 1;
m_large_y = (float)m_text_delta_y; //==
m_small_y = m_large_y / (float)10;//1/10

//设定内部扩大率
if( m_max_value_y == 0 ) m_inzoom_y = (float)1;

//内部扩大率
else m_inzoom_y =  100 / (float)m_max_value_y ;

//右上领域的宽度, 高度  m_resize_x, m_resize_y
m_resize_x = (float)(m_right - m_left);
m_resize_y = (float)(m_bottom - m_top);

//x 界限
if( m_resize_x < 200 ) return(NG);
if( m_shift_val < 30 ) return(NG);
//y 界限
if( m_resize_y < 100 ) return(NG);

//基准长 x
m_zoom_x = ( m_resize_x - (float)60 ) / (float)280;

//基准长 y
m_zoom_y =  m_resize_y  / (float) 120;

//矩形区域描画
```



```
    CPen  Pen(PS_SOLID, 1, RGB(255,255,255) );
    CPen* pOldPen = pDC->SelectObject(&Pen);
    pDC->Rectangle( m_left, m_bottom+1,
                    m_left+(int)m_resize_x+1,
                    m_bottom-(int)m_resize_y );
    pDC->SelectObject(pOldPen);

    return(OK);
}

//x、y 的刻度文字表示
void CDpcmDlg::DrawAxisText( CDC* pDC )
{
    int i;
    float old_posi_y, new_posi_y;
    float old_posi_x, new_posi_x;

    CString str_x, str_y;

    pDC->SetTextAlign( TA_LEFT | TA_BOTTOM );
    pDC->SetTextColor( RGB(0,0,0) );

    if(!m_bDpcmHist)//图像直方图
    {
      //文字拥挤时拉开距离表示
      if( m_text_delta_y != 0 )
      {
        old_posi_y = (float)(m_bottom-20);

        //y 的刻度文字表示
        str_y.Format("%3s", "0 %" );

        // "0%"  文字的微调整
        pDC->TextOut( m_left+17, (int)old_posi_y, str_y, 3 );

        for( i=1; i<(int)(m_max_value_y/m_text_delta_y+1) ; i++ ){
        new_posi_y = (float)m_bottom-(float)20-(m_text_delta_y*float)i*
```



```
  m_zoom_y*m_inzoom_y);

      //y 轴数字间的宽度
      //m_text_interval_y=16
      if( (old_posi_y - (float)16) > new_posi_y ){

        //y 轴刻度数字
        str_y.Format("%3d", m_text_delta_y*i );
        pDC->TextOut( m_left+3, (int)new_posi_y, str_y, 3 );
        old_posi_y = new_posi_y;
      }
    }
  }

  if( m_text_delta_x != 0 ){
    old_posi_x = (float)(m_left+m_shift_val*2 - 15);

    //x 轴刻度数字
    str_x.Format("%3d", m_text_delta_x*0 );
    pDC->TextOut( (int)old_posi_x, m_bottom-5, str_x, 3 );

    for( i=1; i<(int)(m_max_value_x/m_text_delta_x+1) ; i++ ){
      new_posi_x = (float)m_left+(float)m_shift_val*2
       +(m_text_delta_x*(float)i*m_zoom_x*m_inzoom_x)-(float)15;

//x 轴文字间隔
      if(  (new_posi_x - old_posi_x) > (float)28){

        //x 轴刻度数值
        str_x.Format("%3d", m_text_delta_x*i );
        pDC->TextOut( (int)new_posi_x, m_bottom-5, str_x, 3 );
        old_posi_x = new_posi_x;
      }
    }
  }
}
else
```



```
    {
    //DPCM 直方图

      int num;
       old_posi_x = (float)(m_left+m_shift_val*2 - 15);

      //x 轴刻度数字
      str_x.Format("%3d", -256);
      pDC->TextOut( (int)old_posi_x, m_bottom-5, str_x, 4 );

      for( i=1; i<(int)(m_max_value_x/m_text_delta_x+1) ; i++ ){
        new_posi_x = (float)m_left+(float)m_shift_val*2
            +(m_text_delta_x*(float)i*m_zoom_x*m_inzoom_x)-(float)15;

      //x 轴文字间隔
      if(  (new_posi_x - old_posi_x) > (float)28){

         //x 轴刻度数值
         str_x.Format("%3d", m_text_delta_x*i-256 );
         if(m_text_delta_x*i-256 >= 0 ) num = 3;
         else num = 4;
         pDC->TextOut( (int)new_posi_x, m_bottom-5, str_x, num );
         old_posi_x = new_posi_x;
        }
      }
    }
}
void CDpcmDlg::DrawAxisLine(CDC* pDC)
{
    //x 轴描画
    DrawAxisXLine(pDC);
    if(!m_bDpcmHist)
    {
      //y 轴描画
      DrawAxisYLine(pDC);

      //辅助线描画
```



```
        DrawSubLine(pDC);
    }
}

//x 轴描画
void CDpcmDlg::DrawAxisXLine(CDC* pDC)
{
    int i;
    CString str_x;

    //线的幅宽         1
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 坐标
    //x 轴
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*m_zoom_x*
    m_inzoom_x), m_bottom-m_shift_val);

    if(m_bDpcmHist)//DPCM 直方图
    {
      //小刻度
      for( i=0 ; i<(int)(m_max_value_x/m_small_x+1); i++ ){

        //x 轴刻度
        pDC->MoveTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
        pDC->LineTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+2);
    }

      //大刻度
      for( i=0 ; i<(int)(m_max_value_x/m_large_x+1); i++ ){
      //x 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
```



```
      pDC->LineTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
      m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+4);
    }
  }
    else
  {
    //图像直方图
    //小刻度
    for( i=0 ; i<(int)(m_max_value_x/m_small_x+1); i++ ){

          //x 轴刻度
          pDC->MoveTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
          m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
          pDC->LineTo(m_left+m_shift_val*2+(int)(m_small_x*(float)i*
          m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+2);
    }

      //大刻度
      for( i=0 ; i<(int)(m_max_value_x/m_large_x+1); i++ ){

        //x 轴刻度
        pDC->MoveTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val);
        pDC->LineTo(m_left+m_shift_val*2+(int)(m_large_x*(float)i*
        m_zoom_x*m_inzoom_x), m_bottom-m_shift_val+4);
      }
    }
    pDC->SelectObject(pOldPen);
}

//y 轴描画
void CDpcmDlg::DrawAxisYLine(CDC* pDC)
{
    int i;
    CString str_y;

//线的幅宽
```



```
    CPen  Pen(PS_SOLID, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //y 坐标
    //y 轴左
    pDC->MoveTo(m_left+m_shift_val*2,
                m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2,
                m_bottom-m_shift_val-(int)((float)m_max_value_y*m_zoom_y*m_inzoom_y));

      //小刻度
      for( i=0 ; i<(int)(m_max_value_y/m_small_y+1) ; i++ ){
      //y 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2 ,
m_bottom-m_shift_val-(int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));

      pDC->LineTo(m_left+m_shift_val*2-2,
m_bottom-m_shift_val-(int)(m_small_y*(float)i*m_zoom_y*m_inzoom_y));
      }

      //大刻度
      for( i=0 ; i<(int)(m_max_value_y/m_large_y+1) ; i++ ){

        //y 轴刻度
      pDC->MoveTo(m_left+m_shift_val*2,
 m_bottom-m_shift_val-(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y));
      pDC->LineTo(m_left+m_shift_val*2-4,
 m_bottom-m_shift_val-(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y));
      }

      pDC->SelectObject(pOldPen);
 }

 //辅助线描画
 void CDpcmDlg::DrawSubLine(CDC* pDC)
 {
   int st_x, st_y, end_x, end_y, i;
```



```
    //线的幅宽
    CPen  Pen(PS_DOT, 1, RGB(0,0,0) );
    CPen* pOldPen = pDC->SelectObject(&Pen);

    //x 轴
    pDC->MoveTo(m_left+m_shift_val*2, m_bottom-m_shift_val);
    pDC->LineTo(m_left+m_shift_val*2+(int)(m_max_value_x*m_zoom_x*
m_inzoom_x), m_bottom-m_shift_val);
    for( i=1 ; i<6 ; i++ ){
      st_x = m_left+m_shift_val*2;
      st_y = m_bottom-m_shift_val-
(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y);
      end_x = m_left+m_shift_val*2+
(int)(m_max_value_x*m_zoon_x*m_inzoom_x);
      end_y = m_bottom-m_shift_val-
(int)(m_large_y*(float)i*m_zoom_y*m_inzoom_y);
      pDC->MoveTo( st_x,  st_y);
      pDC->LineTo(end_x, end_y);
 }
   pDC->SelectObject(pOldPen);
}

//直方图描画
void CDpcmDlg::DrawGrayValueMono( CDC* pDC )
{
   int i, st_x, st_gray;
   CPen  PenD(PS_SOLID, 1, RGB(255,0,0) );
   CPen* pOldPen = pDC->SelectObject(&PenD);

   UpdateData(TRUE);

   if(m_bDpcmHist)
   {
     //DPCM 直方图
     for( i=0 ; i<512 ; i++ ){
     pDC->SelectObject(&PenD) ;
```



```
      st_x
      m_left+m_shift_val*2+(int)((float)i*m_zoom_x*m_inzoom_x);
      st_gray = (int)(m_disp_array_dpcm[i]*m_zoom_y*m_inzoom_y);
      pDC >MoveTo(st_x, m_bottom-m_shift_val) ;
      pDC >LineTo(st_x, m_bottom-m_shift_val - st_gray) ;
    }
  }
  else
  {
    //图像直方图
    for( i-0 ; i<=255 ; i++ ){
      pDC->SelectObject(&PenD) ;
      st_x
      m_left+m_shift_val*2+(int)((float)i*m_zoom_x*m_inzoom_x);
      st_gray = (int)(m_disp_array[i]*m_zoom_y*m_inzoom_y);
      pDC->MoveTo(st_x, m_bottom-m_shift_val) ;
      pDC->LineTo(st_x, m_bottom-m_shift_val - st_gray) ;
      }
    }
    PenD.DeleteObject() ;
    pDC->SelectObject(pOldPen);
}

void CDpcmDlg::OnDpcmVlc()
{
    //压缩码大小
    int vsize;

    //压缩率
    float ratio;

    //图像符号化
    vsize = ::Dpcm_vlcode(m_pImage_in, m_xsize, m_ysize, m_pImage_buf);

    //表示符号数据
    ::Disp_image(m_pImage_buf);
```



```
    //画面更新
    m_pParent->Invalidate();

    //计算压缩率
    ratio = (float)1 - (float)vsize / (float)(m_xsize*m_ysize);

    //表示压缩比率
    CString str;
    str.Format("压缩了  %.2f%%", ratio*100);
    AfxMessageBox(str);
}

void CDpcmDlg::OnDpcmOrigin()
{
    //表示原图像
    ::Disp_image(m_pImage_in);

    //画面更新
    m_pParent->Invalidate();
}

void CDpcmDlg::OnDpcmVlcImage()
{
    //符号图像化
    ::Idpcm_vlcode(m_pImage_buf, m_pImage_out, m_xsize, m_ysize);

    //表示恢复图像
    ::Disp_image(m_pImage_out);

    //画面更新
    m_pParent->Invalidate();

}
```

附 1.11.2　图像压缩头文件

```
#if !defined(AFX_DPCMDLG_H__8D8CB568_A16F_42DB_B8F4_A8D559391A80__INCLUDED_)
#define AFX_DPCMDLG_H__8D8C8568_A16F_42DB_B8F4_A8D559391A80__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DpcmDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDpcmDlg dialog

class CDpcmDlg : public CDialog
{
// Construction
public:
   CDpcmDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
   //{{AFX_DATA(CDpcmDlg)
   enum { IDD = IDD_DPCM_DIALOG };
     // NOTE: the ClassWizard will add data members here
   //}}AFX_DATA

// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CDpcmDlg)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   virtual void PostNcDestroy();
   //}}AFX_VIRTUAL

   //父亲窗口句柄
   CWnd   *m_pParent;

   //输入图像指针
   BYTE   *m_pImage_in;

   //输出图像指针
```



```
BYTE   *m_pImage_out;

//输出图像指针（码数列）
BYTE   *m_pImage_buf;

//直方图图像指针
BYTE   *m_pImage_hist;

//图像宽度
int     m_xsize;

//图像高度
int     m_ysize;
//直方图的区域左侧
int    m_left;

//直方图的区域下方
int    m_bottom;

//直方图的区域右侧
int    m_right;

//直方图的区域上方
int    m_top;

//描画移动量 30
int     m_shift_val;

//x 轴刻度的最大值
int    m_max_value_x;

//y 轴刻度的最大值
int    m_max_value_y;

//x 轴文字刻度的变化量
int    m_text_delta_x;
```



```
//y 轴文字刻度的变化量
int    m_text_delta_y;

//x 轴小刻度的印记
float m_small_x;

//x 轴大刻度的印记
float m_large_x;

//y 轴小刻度的印记
float m_small_y;

//y 轴大刻度的印记
float m_large_y;

//直方图配列
long    m_hist[256];

//百分比表示直方图配列
float m_disp_array[256];

//最大百分比
float m_max_percent;

//DPCM 直方图配列
long  m_hist_dpcm[512];

//百分比表示 DPCM 直方图配列
float   m_disp_array_dpcm[512];

//窗口领域宽度
float   m_resize_x;

//窗口领域高度
float   m_resize_y;

//扩大率
```



```
float   m_zoom_x;

//扩大率
float   m_zoom_y;

//内部扩大率，以基准高度的 n 倍表示
float   m_inzoom_y;

//以基准宽度的 n 倍表示
float m_inzoom_x;

//轴上文字的表示、非表示
BOOL     m_bAxisText;
BOOL     m_create;_
BOOL     m_bDpcmHist;

//直方图的全体描画
void DispHistogram();

//参数、描画区域设定
short DrawHistArea(CDC* pDC);

//x、y 轴文字的表示
void DrawAxisText(CDC* pDC);

//x、y 轴描画
void DrawAxisLine(CDC* pDC);

//x 轴描画
void DrawAxisXLine(CDC* pDC);

//y 轴描画
void DrawAxisYLine(CDC* pDC);

//辅助线描画
void DrawSubLine(CDC* pDC);
```



```
    //直方图描画
    void DrawGrayValueMono( CDC* pDC );

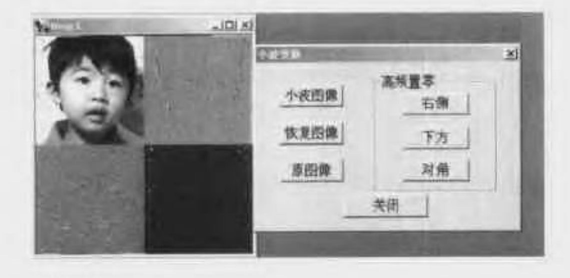
// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CDpcmDlg)
    afx_msg void OnDpcmHist();
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg void OnDpcmImageHist();
    afx_msg void OnDpcmVlc();
    afx_msg void OnDpcmVlcImage();_
    afx_msg void OnDpcmOrigin();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //!defined(AFX_DPCMDLG_H__8D8C8568_A16F_42DB_B8F4_A8D559391A80__INCLUDED_)
```

附 1.12　小波变换



附图 12



附 1.12.1 小波变换源程序

```
// WaveletDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Imgc.h"
#include "WaveletDlg.h"
#include "Global.h"
#include "BaseList.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CWaveletDlg dialog


CWaveletDlg::CWaveletDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CWaveletDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CWaveletDlg)
     // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT

//保存表示画面句柄
   m_pParent = pParent;

   //初始化
   m_pimage_in = NULL;
   m_pimage_out = NULL;
   m_s1 = NULL;
   m_w1v = NULL;
   m_w1h = NULL;
```



```
    m_wld = NULL;
}


void CWaveletDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CWaveletDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CWaveletDlg, CDialog)
    //{{AFX_MSG_MAP(CWaveletDlg)
    ON_BN_CLICKED(IDC_WAVELET_TRANS, OnWaveletTrans)_
    ON_BN_CLICKED(IDC_WAVELET_ITRANS, OnWaveletItrans)
    ON_BN_CLICKED(IDC_WAVELET_FILTER_H, OnWaveletFilterH)
    ON_BN_CLICKED(IDC_WAVELET_FILTER_D, OnWaveletFilterD)
    ON_BN_CLICKED(IDC_WAVELET_FILTER_V, OnWaveletFilterV)
    ON_BN_CLICKED(IDC_WAVELET_ORIGIN, OnWaveletOrigin)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CWaveletDlg message handlers

BOOL CWaveletDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    //获取图像尺寸
    m_xsize = ::GetXSize();
    m_ysize = ::GetYSize();

    //分配内存
```



```
    m_pimage_in = new BYTE[m_xsize*m_ysize];
    m_pimage_out = new BYTE[m_xsize*m_ysize];
    m_s1 = new double[(m_xsize/2)*(m_ysize/2)];
    m_w1v = new double[(m_xsize/2)*(m_ysize/2)];
    m_w1h = new double[(m_xsize/2)*(m_ysize/2)];
    m_w1d = new double[(m_xsize/2)*(m_ysize/2)];

    //读入图像数据
    ::ReadImageData(m_pimage_in);

    //二维小波变换
    ::Wavelet2d (m_pimage_in, m_xsize, m_ysize, m_s1, m_w1v, m_w1h, m_w1d);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CWaveletDlg::OnCancel()
{
    // TODO: Add extra cleanup here
    //释放内存
    delete [] m_pimage_in;
    delete [] m_pimage_out;
    delete [] m_s1;
    delete [] m_w1v;
    delete [] m_w1h;
    delete [] m_w1d;

    CDialog::OnCancel();
}

void CWaveletDlg::OnWaveletTrans()
{
    // TODO: Add your control notification handler code here

    //二维小波变换
    ::Wavelet2d (m_pimage_in, m_xsize, m_ysize, m_s1, m_w1v, m_w1h, m_w1d);
```

```
    //小波信号图像化
    ::Wavelet2d_image (m_s1, m_w1v, m_w1h, m_w1d, m_pimage_out, m_xsize, m_ysize);

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();

}

void CWaveletDlg::OnWaveletItrans()
{
    //二维逆小波变换
    ::Iwavelet2d (m_s1, m_w1v, m_w1h, m_w1d, m_pimage_out, m_xsize, m_ysize );

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();

}

void CWaveletDlg::OnWaveletFilterV()
{

    int i, j;

    //右侧部分置零
    for(j = 0; j < m_ysize/2; j++ )
    {
      for(i = 0; i < m_xsize/2; i++ )
      {
        *(m_w1v + j*m_xsize/2 + i) = 0.0;
        *(m_pimage_out + j*m_xsize + i+m_xsize/2) = 0;
```



```
      }
    }

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();
}

void CWaveletDlg::OnWaveletFilterH()
{
    int i, j;

    //下方部分置零
    for(j = 0; j < m_ysize/2; j++ )
    {
      for(i = 0; i < m_xsize/2; i++ )
      {
        *(m_wih + j*m_xsize/2 + i) = 0.0;
        *(m_pimage_out + (j+m_ysize/2)*m_xsize + i) = 0;
      }
    }

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();

}

void CWaveletDlg::OnWaveletFilterD()
{
    int i, j;
```



```
    //对角部分置零
    for(j = 0; j < m_ysize/2; j++ )
    {
      for(i = 0; i < m_xsize/2; i++ )
      {
        *(m_w1d + j*m_xsize/2 + i) = 0.0;
        *(m_pimage_out + (j+m_ysize/2)*m_xsize + i+m_xsize/2) = 0;
      }
    }

    //表示处理结果
    ::Disp_image(m_pimage_out);

    //更新画面
    m_pParent->Invalidate();_
}

void CWaveletDlg::OnWaveletOrigin()
{

    //表示原图像
    ::Disp_image(m_pimage_in);

    //更新画面
    m_pParent->Invalidate();
}
```

附 1.12.2 小波变换头文件

```
#if !defined(AFX_WAVELETDLG_H__E2FABAAC_374F_4C9A_99BC_302B5F9B4FAD__INCLUDED_)
#define AFX_WAVELETDLG_H__E2FABAAC_374F_4C9A_99BC_302B5F9B4FAD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// WaveletDlg.h : header file
//
```



```
//////////////////////////////////////////////////////////////////////////
// CWaveletDlg dialog

class CWaveletDlg : public CDialog
{
// Construction
public:
   CWaveletDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
   //{{AFX_DATA(CWaveletDlg)
   enum { IDD = IDD_WAVELET_DIALOG };
     // NOTE: the ClassWizard will add data members here
   //}}AFX_DATA

   //父亲窗口句柄
   CWnd* m_pParent;

   //输入图像指针
   BYTE *m_pimage_in;

   //输出图像指针
   BYTE *m_pimage_out;

   //小波信号，左上角
   double *m_s1;

   //小波信号，右上角
   double *m_w1v;

   //小波信号，左下角
   double *m_w1h;

   //小波信号，右下角
   double *m_w1d;

   //图像宽度
```



```
    int m_xsize;

    //图像高度
    int m_ysize;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWaveletDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CWaveletDlg)
    virtual BOOL OnInitDialog();
    virtual void OnCancel();
    afx_msg void OnWaveletTrans();
    afx_msg void OnWaveletItrans();
    afx_msg void OnWaveletFilterH();
    afx_msg void OnWaveletFilterD();
    afx_msg void OnWaveletFilterV();
    afx_msg void OnWaveletOrigin();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif
//!defined(AFX_WAVELETDLG_H__E2FABAAC_374F_4C9A_99BC_302B5F9B4FAD__INCLUDED_)
```

